UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D'INFORMÀTICA DE BARCELONA

Design of an environment for solving pseudo-boolean optimization problems

Author:
Marc BENEDÍ

Supervisor:
Dr. Jordi CORTADELLA

Dissertation

June 26, 2018
Geneva, Switzerland

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstract

Facultat Informàtica de Barcelona
Department of Computer Science

Computer Science Degree

Facultat d'informàtica de Barcelona

by Marc BENEDÍ

This dissertation addresses several approaches with the common goal of reducing the time required to solve Pseudo-Boolean minimisation problems. A C++ library has been developed which allows representing Pseudo-Boolean minimisation problems smoothly. The problem entered for the user is encoded into a CNF using PBLib. Two different algorithms are used to search for the minimum value for the cost function: Linear search and Binary search. Finally, two timeout strategies are implemented: General timeout and Simple timeout which stops the solver and returns the last minimum value found.

Glossary

Abbreviation	Description of the abbreviation
BF	**B**oolean **F**ormula
CNF	**C**onjunctive **N**ormal **F**orm
DNF	**D**isjunctive **N**ormal **F**orm
PB	**P**seudo-**B**oolean
PBF	**P**seudo-**B**oolean **F**ormula
SAT	Boolean **sat**isfiability problem
PB-Min	**P**seudo-**B**oolean **Min**imisation
BF	**B**oolean **F**ormula
NP	**N**ondeterministic **P**olynomial time
BDD	**B**inary **D**ecision **D**iagram
AI	**A**rtificial **I**ntelligence
TDD	**T**est-**D**riven **D**evelopment

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Context

1.1 Introduction

Technology's world is evolving very fast. Every year new tools, frameworks[1], programming languages,... are released. Some of them are living their gold's era, such as blockchain and machine learning, while others are forgotten.
Because of its fast evolution, this makes hard for universities decide what to teach: the tools used by companies or the fundaments of informatics. This project works with the fundaments of informatics and logic which will neither change nor be forgotten.

In this project, we will work with Pseudo-Boolean Minimisation widely used by many technologies such as Artifical Intelligence, Planners, Computer Vision, Circuit design, ...

1.2 Context

However, before being able to explain the main problem which this project is about, Pseudo-Boolean Minimisation, it is necessary to do a quick introduction to a much broader topic.

Boolean satisfiability problems (SAT from now on) is the problem of finding a model for a Boolean Formula (BF from now on). In other words, it is the result of evaluating the BF after replacing its variables for *true* or *false*.
SAT is widely used in Computer Science because it was the first problem proved to be NP-Complete which allowed reducing many NP problems to it[2].

1.2.1 What is a Boolean Formula?

In propositional logic, a *BF* is defined as follows[3]:
Let P be a set of predicate symbols like $p, q, r, ...$

- All predicate symbol of P is a formula.
- If F and G are formulae, then $(F \land G)$ and $(F \lor G)$ are formulae too.
- If F is a formula, then $(\neg F)$ is a formula.
- Nothing else is a formula.

[1]A software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. Wikipedia

This representation has some limitations because it can only express properties which are *true* or *false*.

For example, given the variables a, b, c, $a \vee b \wedge c$ is a BF. The interpretation $a = b = c = false$ is not a model because the formula's evaluation is $false$ whereas the interpretation $a = b = true$ and $c = true$ is a model.

This representation has many uses for example in propositional logic, i.e. specification and verification of circuits and systems, programming languages, databases, ...

1.2.2 What are Pseudo-Boolean Formulae?

Pseudo-Boolean Formulas are functions of the form $f : B^n \to \mathbb{R}$. For example, the following formula is a *Pseudo-Boolean Formula* (PBF from now on): $3x + 5y$. Therefore, *BF* are a special case of *PBF* where the domain is $d = \{0, 1\}$.

1.2.3 What are Pseudo-Boolean constraints

Pseudo-Boolean constraints are derived from PBF. They are expressions of the form $PBF \leq k$, where $k \in \mathbb{Z}$.There are used to express restrictions because to satisfy them it is necessary to find an assignment which its evaluation is $\leq k$.
For example, given the variables $a, b, c \in \mathbb{B}$, and the PBC $3a + 5b + c \leq 7$, the interpretation $a = b = c = true$ does not satisfy the constraint because $9 \not\leq 7$, whereas the interpretation $a = b = c = false$ does satisfy it.

1.2.4 Pseudo-Boolean minimisation

Pseudo-boolean optimisation is a linear programming problem:

Linear programming (LP, also called linear optimization) is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships.

Linear programs are problems that can be expressed in canonical formas

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq \mathbf{0} \end{array}$$

FIGURE 1.1: Linear programming canonical form representation

where x represents the vector of variables (to be determined), c and b are vectors of (known) coefficients, A is a (known) matrix of coefficients, and T is the matrix transpose. The expression to be maximised or minimised is called the objective function ($c^T x$ in this case). The inequalities $Ax \leq b$ and $x \geq 0$ are the constraints which specify a convex polytope over which the objective function is to be optimized.

https://en.wikipedia.org/wiki/Linear_programming

Pseudo-boolean minimisation is a particular case where the objective function (also called cost function) is minimised, the variables are boolean $\mathbb{B} \in \{0,1\}$ and the variable's coefficients are integers $\mathbb{Z}$.
In other words, a pseudo-boolean minimisation problem has two components:

- A set of pseudo-boolean constraints
  of the form $\sum_{i=1}^{n} x_i w_i \leq k$, where $w_i, k \in \mathbb{I}$ and $x_i \in \{0,1\}$
- A cost function

The goal is to minimise the cost function $min(a_1 x1 + ... + a_n x_n)$. Find an assignment for the problem's variables in a way that all the constraints are satisfied and the evaluation of the cost function is minimum.
The value m is minimum, if $m - 1$ is unsatisfiable and m satisfiable.

An example of a problem which can be solved with Pseudo-Boolean minimisation is the Knapsack problem[2]. The problem consists of the following: Given a set of items, each one with a value and a weight, and a knapsack with a maximum capacity, the goal is to select some objects to put inside the knapsack in a way that the weight is not bigger than the knapsack's capacity and the total value is maximised.

The variables for this problem, given n objects, are:

$$o_1, o_2, \ldots, o_n \text{for the objects}$$

$$w_1, w_2, \ldots, w_n \text{for the weights}$$

$$v_1, v_2, \ldots, v_n \text{for the values}$$

There is only one constraint which represent the maximum capacity of the knapsack problem:

$$w_0 o_0 + w_1 o_1 + \ldots w_n o_n \leq knapsack's\ capacity$$

And the cost function:

$$v_0 o_0 + v_1 o_1 + \ldots v_n o_n$$

There is a big research in this field, more specifically in encoding PBF into CNF. In this paper, Hölldobler, Manthey, Steinke[6], some relevant PBF into SAT encodings are explained and a new one is proposed. One of the authors of this paper, Steinke, is also the author of PBLib.

1.3 Background

During the past semester (Q1 2017/2018), under the supervision of Dr Jordi Cortadella, the author had been developing a C++ library.
This tool allows the users to represent *BF* in a C++ program in an intuitive way, do operations between them and convert them into *Binary Decision Diagrams* (BDD from now on). However, the main functionality of this library is the conversion from a *BF* to *CNF*.
As previously explained, *CNF* is a particular type of a *BF*, a conjunction of disjunctions. *CNF* is an essential format because it is the standard input for *SAT Solvers*

[2]https://en.wikipedia.org/wiki/Knapsack_problem

[A.1].
As shown in this paper, *Mitchell, Selman, and Levesque [9]*, there is a correlation between the number of variables, the number of clauses and the hardness of solving the *CNF*.

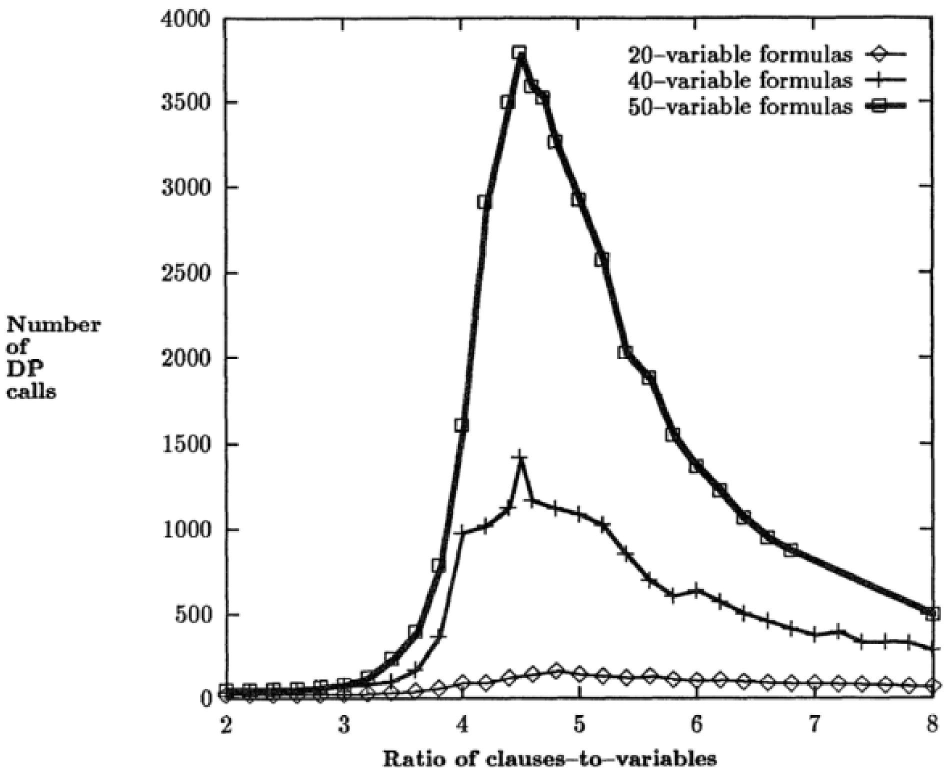


FIGURE 1.2: Median number of recursive DP calls for Random 3-SAT formulas, as a function of the ratio of clauses-to-variables. Extracted from Mitchell, Selman, and Levesque[9]

Therefore, an improvement of the input *CNF* of the *SAT Solver* can reduce a lot the hardness of the problem.
This is the primary goal of the library, trying to reduce the size of the final *CNF* resulting from applying different converting methods on the original BF.

1.4 State-of-the-art

Nowadays there is a well known C++ library called PBLib developed by Peter Steinke. PBLib allows its users to represent Pseudo-Boolean constraints and encode them into a CNF. It possesses a big repertory of encodings which have been proposed in the past in different papers[12].

At most one	At most K	PB
sequential[11]*	BDD[7, 4]**	BDD
bimander[6]	cardinality networks[1]	adder networks
commander[8]	adder networks[4]	watchdog[10]
k-product[3]	*todo: perfect hashing*[12]	sorting networks[4]
binary[2]		binary merge[9]
pairwaise		sequential weight counter[5]
nested		

* equivalent to BDD, latter and regular encoding
** equivalent to sequential counter

FIGURE 1.3: PBLib implemented encodings
Extracted from Steinke[12]

Apart from having these encodings, PBLib applies the best one for the given Pseudo-Boolean constraint in a way that provides the most efficient translation.

In Chapter3 - Development [3], PBLib is discussed as a tool for this project. In a nutshell, PBLib is an excellent tool developed by professionals, and it will be hard to try to do the same with the same quality. For these reasons, and some others explained in more detail in Chapter 3 [3], PBLib has been incorporated in this project as a translator of Pseudo-Boolean constraints to CNF.
One of the downsides of PBLib is the interface which is not very user-friendly. As can be seen in Chapter 2 - Objectives [2], one of the objectives of this project is offering a better interface which means that a layer between the user and PBLib will be added to simplify how the users declare pseudo-boolean constraints.

1.5 Motivations

Informatics Logic is taught in this[3] faculty. In this course, the author realised how relevant is *logic* through its lecturer, Dr Robert Nieuwenhuis, and its activities.
In the first coursework, we had to code a *SAT Solver* which used *Unit Propagation*. With this activity, the author comprehended how hard and substantial is the study of *logic* and all its context. For example, how *logic* is used in Artificial Intelligence and Planners.

When the time of deciding the *TFG* arrived, he contacted his actual supervisor, Dr Jordi Cortadella, and he proposed him some topics and ideas. Finally, they agreed on doing this project.

The motivation for this project is trying to deepen into the topic and contribute on it.
Also during the studies, with the transversal competencies, there has been an intention to teach students about sustainability and its importance.
As previously told, the subject of this project is widely used in many other areas which have a big footprint because of electricity consumption. This is also a motivation for this project because reducing the execution time will become in a decrease of electricity spent and that will be beneficial for the climate.

[3]Facultat Informàtica de Barcelona

1.6 Stakeholders

In this section, the Stakeholders of the project are defined. Stakeholders are entities which are affected, directly or indirectly, by the solution developed in this project.

1.6.1 Target audience

This tools targets all the entities (researchers, companies, . . .) which work with *PB minimisation* and use *SAT Solvers*.

1.6.2 Users

The users will be C++ programmers due this tool is developed in this language.

1.6.3 Beneficiaries

All those entities which work with *PB minimisation*. For example AI, SAT Solvers, Planners, . . .

Chapter 2

Project Scope

2.1 Project Formulation

As previously mentioned [1], this project is an extension of a previous C++ library. The main goal of this project is to improve the time required to solve *Pseudo-Boolean minimisation* problems.
To achieve this goal the following objectives have been established.

2.1.1 General objectives

Pseudo-Boolean minimisation

For the problems formed by some *PB Constraints* $c_1x_1 + c_2x_2 + \ldots + c_nx_n \leq k$ and a cost function , the goal is to find and assignment for $\{x_1, x_2, \ldots, x_n\}$ in a way that the cost function is minimised.

$$min(a_1x1 + ... + a_nx_n)$$

Previously [1], it has been explained that these types of problems are *NP-Hard*. This project will try to reduce the required time to solve these problems through two approaches:

- Binary search:
  Implement the well known *Binary Search* algorithm to find the minimum value for the cost function.
- Linear search:
  Some *SAT Solvers* can learn and derive new restrictions from previous problems. To take advantage of this ability, it is necessary to implement a *Linear Search* algorithm.

Timeout

For some problems, it is more important to find a solution before a deadline than finding the best possible solution. For instance, a delivery company must have all the routes planned for all trucks before the journey starts, therefore, they care more about having a solution than finding the best one.
For this, a *Timeout strategy* will be implemented in case that a good enough solution has been found or the problem does not seem to have one.

Multi-threading (Optional)

This tool will take advantage of multi-core processors trying to split the problem and solving each part separately.

2.2 Scope

2.2.1 What and how?

To achieve all the general objectives [2] of the project the following stages have been established:

- Analyze, refactor[1] and test the existing code to have a solid base.
- Add the functionality of representing *PBF*.
- Study PBLib library to see which functionalities it has available to work with *minimisation*.
- Implement *minimisation* strategies.
- Study timeout strategies and implement them.
- Study and implement multithreading. (Optional)

2.2.2 Possible obstacles

In this section, the possible obstacles and its solutions are exposed.

Base project

The existing project *Background section* [1] has not been developed following an adequate methodology. This bad practice could be responsible for a poor code quality. Building on top of a program with this characteristics could have terrible consequences because it would produce a lot of bugs and malfunctions hard to solve in the future.
For this reason, it is better to improve the quality of the existing code, because it will avoid problems in the future.

Schedule

Because this is a final degree project, the scope is limited, in this case until June of 2018. Considering the learning stage, analysis, requirements study, and other deviations which could appear, the time available to develop the project could be drastically reduced. Moreover, this project will be developed in an Erasmus stay which makes the planning more laborious. For these reasons, a good and realistic planning are key steps to take advantage of time and reduce contingencies.

PBLib

One of the main requirements of this project, *Pseudo-Boolean minimisation*, is planned to be done with *PBLib* library. This decision could be an obstacle because *PBLib* could not be compatible with the project causing compiling errors and therefore some time would have to be spent solving them. Also, *PBLib* could not have the expected functionalities, in which case a substitute should be found or, even worse, having to implement *PBLib* functionalities which would take too much time.

[1]Code refactoring is the process of restructuring existing computer code—changing the factoring—without changing its external behavior. (more)

Correctness

As explained in *Rigor and Validation* [6], correctness in this project is fundamental because of the context it is in.
Guaranteeing correctness could be hard and take more time than expected. If this happens, formal correctness could be delayed or reduced.

2.3 Methodology and Rigor

Research is a vast process with no clear path between *a* and *b*. For this, it is important to follow some directions. A methodology will provide some guidelines to avoid possible problems, be more efficient and do the project more manageable.

2.3.1 Methodology

The methodology adopted for this project will be Agile[2]. It is important to clarify that this methodology will not be followed strictly but adapted to this particular case where there is only one developer and all the objectives are well defined. The main characteristics followed from Agile in this project will be:

- Short cycles
- TDD (Test-Driven Development)
- Weekly scrums with the supervisor

2.3.2 Tools

In this chapter the development tools will be introduced.

Git

Git will be used in this project as a Version Control System because it allows maintaining tracking of all the changes made (commits), and what is more important, return to them at any time. In addition to this, it enforces a short cycle development (because commits are small units of work) and the developer has to document them which matches perfectly with Agile methodology. GitHub will be the repository service used.

Trello

Trello is a simple and flexible web board which helps to organize tasks and its state. It will be used in this project to manage tasks and priorities.

2.3.3 Communication

Due to the author conditions, who has been studying abroad in an Erasmus program, all the communication has been made through electronic means. The majority of it has been made using e-mail but when necessary video conferences were done.
The minimum communication with the supervisor has been a weekly e-mail report where all the tasks completed during that period were explained. Problems or questions were also exposed if any.

[2]Methodology based on the on the adaptability in front of any change to improve exit possibilities.

2.3.4 Rigour and Validation

Rigor and Validation for this project are relevant.
The surrounding of it, such as *Artificial Intelligence, Planners, Cryptographic Protocols verification, . . .* , are widely used nowadays and have been becoming more popular lately. This means that this project could have a significant repercussion and be used by some professionals. For this, it is important to guarantee the validation and correctness of the project.
During the development, TDD will be used to avoid unnecessary code (possible origin of bugs) and assure the correctness of the implementation. It is also possible to formalize and prove all the operations done by the software.
Finally, the project's supervisor could give some orientation and validate, if necessary, the operations done.

Chapter 3

Development

3.1 Requirement analysis, architecture and debugging

As previously mentioned [2], this stage has been dedicated to four points:

- Environment preparation
- Refactor
- Requirements
- Architecture

3.1.1 Environment configuration

First things first, before being able to start working it was necessary to prepare and set up all the tools needed for the project. Some of them were already configured from the previous project, such as the Control Version System, but some others would need to be configured from scratch or modified.

Makefile

The existing Makefile at that moment was deleted, and a new one was created. The goal of a Makefile is to automatize the build of the software and save time. The Makefile would be modified during the development stages to incorporate the new targets/goals[1].

Google Test

TDD (Test Driven Development) was adopted as part of the methodology for the development stages. For this, a testing framework was required. Not many options were considered because Google Test is highly popular and widely used.
Google Test is explained in more detail in the appendix B.

Adopting and learning a new framework has some costs, but there the trade-off was evident because the time required for learning was smaller than the time required for finding and solving bugs. It is also very relevant that using a testing framework gives confidence in the code because it guarantees that it works.

[1] **https://en.wikipedia.org/wiki/Makefile**.

3.1.2 Refactor

A refactor of the existing code was needed. As it has been previously mentioned, this project has been built onto an existing code which needed to be tested.
Once the Google Test framework was set up, the author started creating Unit Tests for each functionality of the existing code.
Some bugs were found and corrected. Doing this at the beginning of the project was an excellent decision because those bugs would have caused erroneous behaviour painful to track once the project became bigger.

3.1.3 Requirements

For the requirements, a more in-depth look at them was done. At that point, it was time to list them and plan how to achieve them. It was essential to bear in mind that for each iteration there would be a planning substage. Therefore, the requirements mentioned at that point were the global ones which would affect the architecture. It would have been difficult trying to think about all the requirements at the beginning and inefficient because of the Agile methodology. The global requirements for the software were:

- Testability, upgradeability, and modifiability
- Easy to use
- Fast

As the reader can see, the first requirements are software/development related. Those have been very important during the development stages, and they have influenced a lot the architecture.

3.1.4 Architecture

The primary goal of the architecture at this point was to respect the SOLID principle. SOLID [8] stands from:

- Single responsibility principle: A class should have only one responsibility.
- Open/closed principle: Software entities should be open for extension but closed for modification.
- Liskov substitution principle: Objects should be replaceable by instances of their subtypes.
- Interface segregation principle: "many client-specific interfaces are better than one general-purpose interface."
- Dependency inversion principle: The dependencies should depend upon abstractions and not concretions.

3.2 Objective 1: Pseudo-boolean Minimisation

3.2.1 Problem

This iteration addressed some problems: First, make the software easy to use while efficient to work with, and second, find the optimal value of a PBMin problem using different techniques.

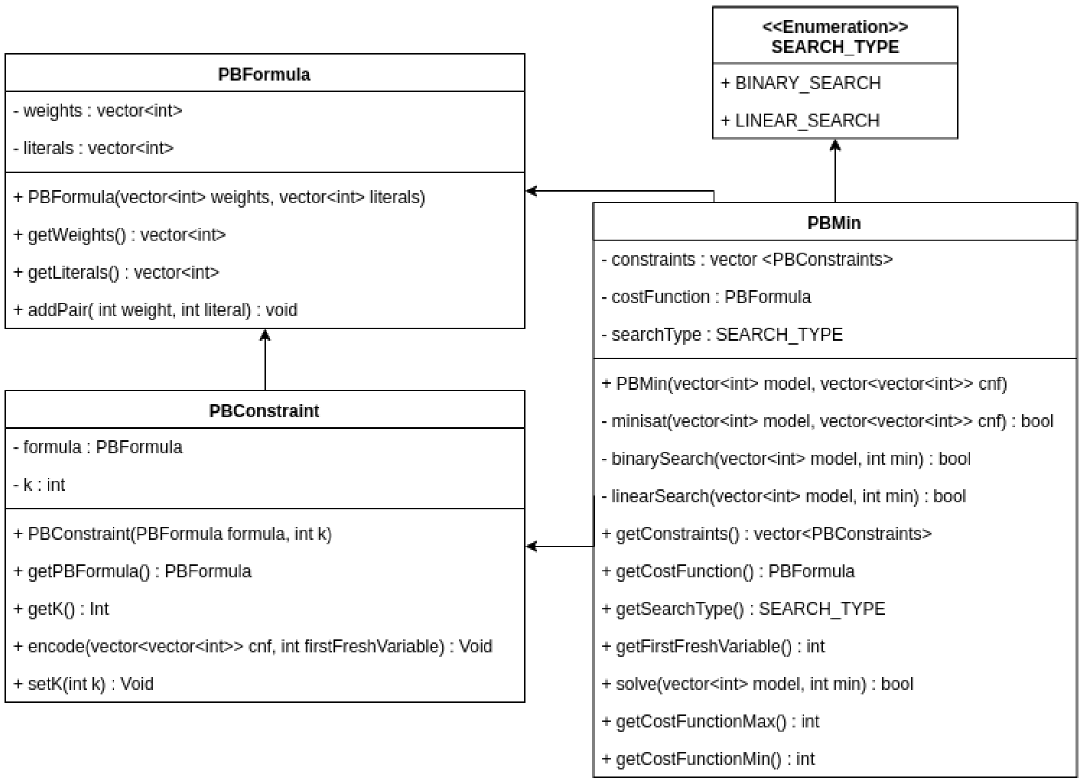
3.2.2 Possible solutions

One solution was creating from scratch the software required to represent PBFormulae and their encoding to CNF or incorporate a popular C++ library, PBLib, into the project.

The first option was quickly dismissed because it would have taken too much time, and the existing alternatives were distinguished and it would have been very hard to compete against them. Therefore, the second option was the chosen one because it would be faster and it would provide a better quality solution.

3.2.3 Planning

First of all, it was required to add PBLib to the tool's set with the following steps: download and install it. Then modify the Makefile to add the new dependence and compile a simple C++ program which used PBLib and saw if it worked.
Once the whole project was compiling and executing without errors, it was time to start designing the architecture for this first iteration:



<<Enumeration>>
SEARCH_TYPE
+ BINARY_SEARCH
+ LINEAR_SEARCH
PBFormula
- weights : vector<int>
- literals : vector<int>
+ PBFormula(vector<int> weights, vector<int> literals)
+ getWeights() : vector<int>
+ getLiterals() : vector<int>
+ addPair(int weight, int literal) : void
PBConstraint
- formula : PBFormula
- k : int
+ PBConstraint(PBFormula formula, int k)
+ getPBFormula() : PBFormula
+ getK() : Int
+ encode(vector<vector<int>> cnf, int firstFreshVariable) : Void
+ setK(int k) : Void
PBMin
- constraints : vector <PBConstraints>
- costFunction : PBFormula
- searchType : SEARCH_TYPE
+ PBMin(vector<int> model, vector<vector<int>> cnf)
- minisat(vector<int> model, vector<vector<int>> cnf) : bool
- binarySearch(vector<int> model, int min) : bool
- linearSearch(vector<int> model, int min) : bool
+ getConstraints() : vector<PBConstraints>
+ getCostFunction() : PBFormula
+ getSearchType() : SEARCH_TYPE
+ getFirstFreshVariable() : int
+ solve(vector<int> model, int min) : bool
+ getCostFunctionMax() : int
+ getCostFunctionMin() : int

FIGURE 3.1: Iteration 1 architecture

PBFormula This class is the one responsible for representing a Pseudo-Boolean Formula. In order to make it more interoperable with PBLib, it adopted the same representation for variables and weights which are int32_t and int64_t respectively.

PBConstraint A Pseudo-Boolean Constraint is represented as a PBFormula and a boundary.
The function *encode()* is responsible for converting the Pseudo-Boolean Constraints

into a CNF. This encoding is the part done with PBLib.
Note that there is no way of specifying the relational operator of the constraint because it will always be $\leq$. Other constraints can be easily converted into this type as specified in this paper [1].

PBMin This class is responsible for representing a Pseudo-Boolean Minimisation problem. It is formed by a vector of PBConstraint, the cost function which is a PBFormula and the search type which is defined by the enum *SEARCH_TYPE*.
This enum has two values, *BINARY_SEARCH* and *LINEAR_SEARCH*, which are the search strategies specified in the objectives of the project.

This class has the following methods:

- *minisat()* is responsible for calling the solver, Minisat, and get the model back if it exists.
- *binarySearch()* is responsible for executing the Binary Search algorithm to find the optimal value for the problem.
- *linearSearch()* is responsible for executing the Linear Search algorithm to find the optimal value for the problem.
- *getFirstFreshVarialbe()* returns a value named *first fresh variable* which is required by PBLib in order to encode the constraints into a CNF.
- *solve()* is called by the user once he/she wants to solve the problem.
- *getCostFunctionMax()* returns the maximum possible value for the cost function.
- *getCostFunctionMin()* returns the minimum possible value for the cost function.

As the reader may have noticed, this contradicts the initial intention which was that this class should be only responsible for representing a Pseudo-Boolean Minimisation problem. The methods listed above show that the class is also responsible for calling Minisat, implementing the search strategies and, as a consequence of this, calling PBLib for encoding the problem into a CNF.

> This error in the class design had a negative impact on the next iteration which required a redesign of the architecture. It will be explained in this section [3.3.3].

3.2.4 Development and TDD

The development started with the leaf class at the hierarchy, as seen in the architecture [3.1], PBFormula.
Following the hierarchy, the next one was PBConstraint and the last one PBMin.

The first implemented method from PBMin was *int32_t PBMin::getFirstFreshVariable()*.
This method returns the next available literal to be used in the future by PBLib. To achieve this behaviour, this method looks at all the literals that are in the PBMin (constraints and cost function) and returns the maximum absolute literal found plus one.
For example, if we have the constraint $3 * 1 + 2 * (-2) \leq 3$, and the cost function

$4 * (-1) + 7 * 2$, the first fresh variable is 3.

The methods *int64_t PBMin::getCostFunctionMax()* and *int64_t PBMin::getCostFunctionMin()* are required to define the boundaries of the search strategies. The optimal value has to be comprehended between these boundaries.
The first part of these two methods does the same: For each literal at the cost function sums the weights where it appears positive and the weights where it appears negated and stores them in *positive[i]* and *negative[i]* where *i* is the literal.
The maximum value for the cost function is that where a literal is set to true if *positive[i]* is bigger than *negative[i]*, and false otherwise . Similarly, the minimum value for the cost function is that where a literal is set to true if *negative[i]* is bigger than *positive[i]*, and false otherwise.

For example, given the cost function $-1 * 1 - 3 * (-1) + 7 * 2 - 5 * (-2)$,
positive = {-1,7}
negative = {-3,-5}
Literal 1 is set to *true* because $positive[1] > negative[1]$.
Literal 2 is set to *true* because $positive[2] > negative[2]$.
The maximum value for the cost function is 6.
Literal 1 is set to $false$ because $positive[1] > negative[1]$.
Literal 2 is set to $false$ because $positive[2] > negative[2]$.
The minimum value for the cost function is -8.

3.2.5 Finalization

At this substage, some techniques learned at the Erasmus course, Software Testing, such as Category Partition Testing were applied.
These new tests revealed some bugs in the search strategies implementation for particular values for the algorithms.

Class	*min* is the first value	*min* is the last value	Unsatisfiable
Linear Search	$\overline{x_1} \leq 0$ x_1	$x_1 \leq 0$ x_1	$2x_1 \leq 1$ $2\overline{x_1} \leq 1$ x_1
Binary Search	$\overline{x_1} \leq 0$ $5x_1 + 5x_2$	$\overline{x_1} \leq 0$ $3x_1 + 7x_2$	$2x_1 \leq 1$ $2\overline{x_1} \leq 1$ $x1$

TABLE 3.1: Category partition testing for *Linear search* and *Binary search*

3.3 Objective 2: Timeout

3.3.1 Problem

It has been previously explained that SAT is an NP-Complete problem which means that there is no known algorithm which can solve it in polynomial time. In other words, it can take a lot of time to solve this type of problems.
For example, a CNF of 300 variables and 1200 clauses takes 50 seconds to be solved.

Some users need a result before certain time. For instance, a delivery company needs a plan every morning before 08:00 AM for their trucks,packages and drivers. Therefore some users need a result before a certain time even if it is not the optimal one.

In order that the software was able to work with this new feature, it needed a new parameter from the user and timeout strategies.

3.3.2 Possible solutions

Two different timeout strategies were considered:

- A maximum number of seconds for each call to the solver.
- A maximum number of seconds for the whole problem.

Both options were considered useful, and hence they became a goal for the development substage.

3.3.3 Planning

Some research was done about timeout strategies for solvers.
In order to implement the asynchronous behaviour two different approaches were considered:

Threads Threads within the same process can communicate using shared memory. Threads share the same memory space which could be a source of problems if not handled carefully. In that particular case, that was helpful because the thread would execute the solver call while the parent was waiting for the timeout.

Processes Child processes are easier to work with but, in general, they are more expensive. This is because processes have their own code, memory space, files, registers and stack which implies that every time a new child process is created, some of the listed elements need to be copied.

Finally, Threads was the chosen option because of the following reasons:

- They sharer memory within a process which makes communication much more manageable. In this particular project, this allowed the code to be more simple.
- They are less expensive to create because no copy of memory is effectuated.

The timeout was thought to be implemented as a signal sent from the parent thread to the child thread.

What happens when there is a timeout?
The answer depends on the timeout strategy selected for the problem: *GeneralTimeoutSolver* or *SimpleTimeoutSolver*.

In the first case, the flag *timeoutOccurred* is set to *true*, and the last found solution is returned. With the flag the user can know if the timeout occurred and therefore be aware that the solution may not be optimal.

In the second case, the flag *timeoutOccurred* is also set to *true* and the current solver execution is killed. However, the search algorithm will keep running with the next values.

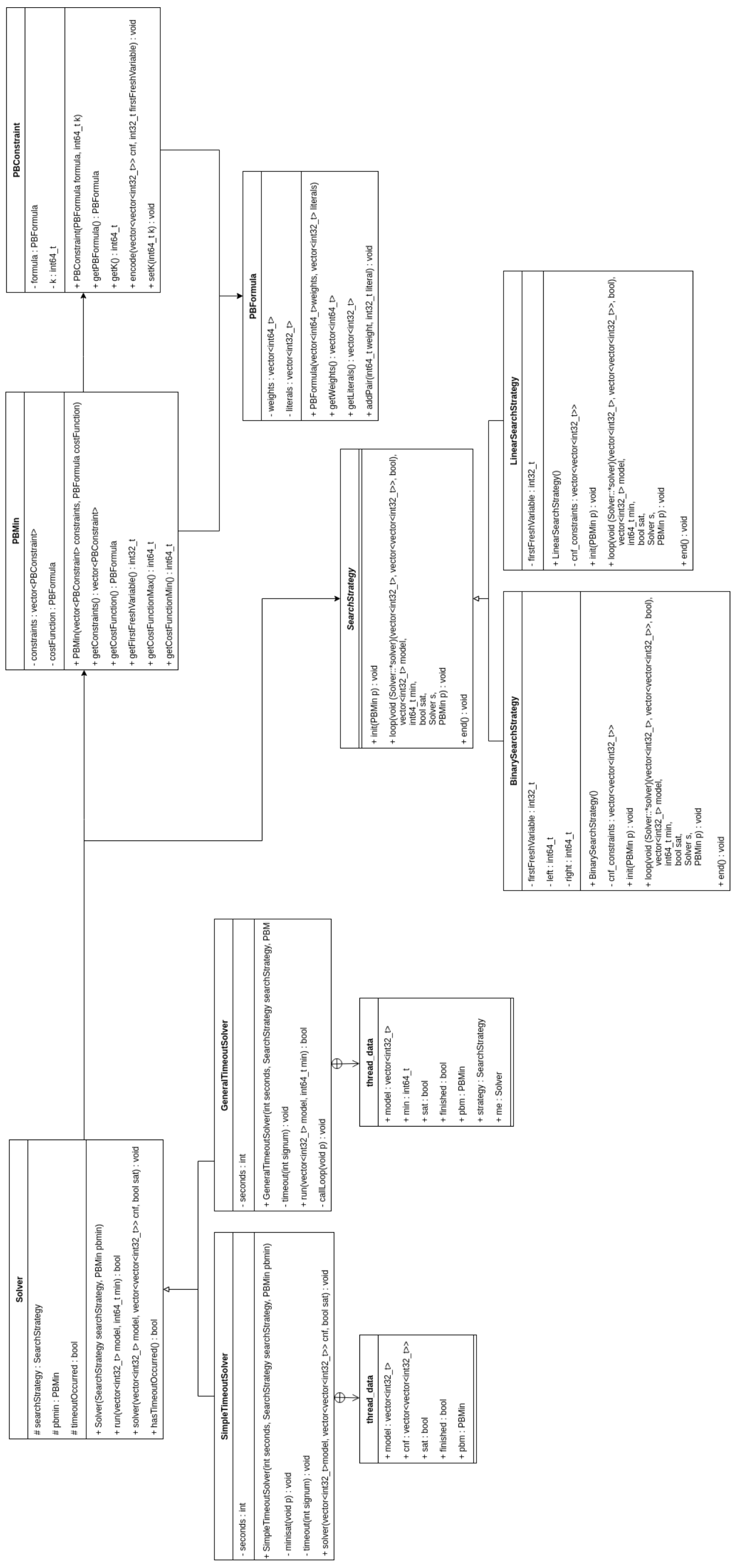

PBConstraint
- formula : PBFormula
- k : int64_t
+ PBConstraint(PBFormula formula, int64_t k)
+ getPBFormula() : PBFormula
+ getK() : int64_t
+ encode(vector<vector<int32_t>> cnf, int32_t firstFreshVariable) : void
+ setK(int64_t k) : void
PBFormula
- weights : vector<int64_t>
- literals : vector<int32_t>
+ PBFormula(vector<int64_t>weights, vector<int32_t> literals)
+ getWeights() : vector<int64_t>
+ getLiterals() : vector<int32_t>
+ addPair(int64_t weight, int32_t literal) : void
PBMin
- constraints : vector<PBConstraint>
- costFunction : PBFormula
+ PBMin(vector<PBConstraint> constraints, PBFormula costFunction)
+ getConstraints() : vector<PBConstraint>
+ getCostFunction() : PBFormula
+ getFirstFreshVariable() : int32_t
+ getCostFunctionMax() : int64_t
+ getCostFunctionMin() : int64_t
SearchStrategy
+ init(PBMin p) : void
+ loop(void (Solver::*solver)(vector<int32_t>, vector<vector<int32_t>>, bool), vector<int32_t> model, int64_t min, bool sat, Solver s, PBMin p) : void
+ end() : void
LinearSearchStrategy
- firstFreshVariable : int32_t
+ LinearSearchStrategy()
- cnf_constraints : vector<vector<int32_t>>
+ init(PBMin p) : void
+ loop(void (Solver::*solver)(vector<int32_t>, vector<vector<int32_t>>, bool), vector<int32_t> model, int64_t min, bool sat, Solver s, PBMin p) : void
+ end() : void
BinarySearchStrategy
- firstFreshVariable : int32_t
- left : int64_t
- right : int64_t
+ BinarySearchStrategy()
- cnf_constraints : vector<vector<int32_t>>
+ init(PBMin p) : void
+ loop(void (Solver::*solver)(vector<int32_t>, vector<vector<int32_t>>, bool), vector<int32_t> model, int64_t min, bool sat, Solver s, PBMin p) : void
+ end() : void
GeneralTimeoutSolver
- seconds : int
+ GeneralTimeoutSolver(int seconds, SearchStrategy searchStrategy, PBM
- timeout(int signum) : void
+ run(vector<int32_t> model, int64_t min) : bool
- callLoop(void p) : void
thread_data
+ model : vector<int32_t>
+ min : int64_t
+ sat : bool
+ finished : bool
+ pbm : PBMin
+ strategy : SearchStrategy
+ me : Solver
Solver
# searchStrategy : SearchStrategy
# pbmin : PBMin
# timeoutOccurred : bool
+ Solver(SearchStrategy searchStrategy, PBMin pbmin)
+ run(vector<int32_t> model, int64_t min) : bool
+ solver(vector<int32_t> model, vector<vector<int32_t>> cnf, bool sat) : void
+ hasTimeoutOccurred() : bool
SimpleTimeoutSolver
- seconds : int
+ SimpleTimeoutSolver(int seconds, SearchStrategy searchStrategy, PBMin pbmin)
- minisat(void p) : void
- timeout(int signum) : void
+ solver(vector<int32_t>model, vector<vector<int32_t>> cnf, bool sat) : void
thread_data
+ model : vector<int32_t>
+ cnf : vector<vector<int32_t>>
+ sat : bool
+ finished : bool
+ pbm : PBMin

FIGURE 3.2: Iteration 2 architecture

As can be seen, the architecture suffered an evolution from the previous iteration. As previously mentioned in Iteration 1, the class PBMin was not respecting the Single responsibility principle because apart from representing the problem, it was also responsible for implementing the search strategies, calling PBLib encodings, ...
That architecture would have resulted in a much complex PBMin class because it would have to be responsible for the timeouts.

For these reasons the following classes were modified or added:

- PBMin : After the refactor, this class is only responsible for representing a Pseudo-boolean minimisation problem which was its original purpose.
- SearchStrategy : One of the functionalities moved outside the PBMin class was the search strategy selection and implementation which needed to be placed into another class. For that reason, a new hierarchy of classes was created: SearchStrategy, BinarySearchStrategy and LinearSearchStrategy. These set of classes have three methods:
  - *init()*'s purpose is to prepare the class for the execution of the search algorithm. For example, encode the PBConstraints into a CNF.
  - *loop()*'s purpose is to execute the search algorithm.
  - *end()* is executed after the loop once the optimal solution is found, there is no solution or a timeout occurred.
- *BinarySearchStrategy* uses the *loop()* method to implement the binary search algorithm.
- *LinearSearchStrategy* uses the *loop()* method to implement the linear search algorithm.
- *Solver* : Another behaviour removed from the PBMin class is the execution of the problem solver. This behaviour was moved to a new class, Solver, which is responsible for calling the SAT solver. This class is the solver without a timeout, that is implemented by its children SimpleTimeoutSolver and GeneralTimeoutSolver. The main methods are:
  - *run()* is the function called by the user once he/she wants to solve the problem.
  - *solve()* is responsible for calling the SAT solver, passing the CNF and getting back the result.
- *SimpleTimeoutSolver* overrides the method *solve()* and creates a thread which calls the solver while it counts for the timeout.
- *GeneralTimeoutSolver* overrides the method *run()* and creates a thread which calls the selected search algorithm while it counts for the timeout.

3.3.4 Development and TDD

The first thing to do was refactor the class PBMin and move all the functionalities previously mentioned to the new classes. This also implied a refactor to the PBMin tests. Like the previous iteration, the development was done selecting classes in a bottom-up way.

The first class created was Solver which does not implement a timeout strategy. Therefore it has the same functionality as the old PBMin: call the search algorithm and the SAT solver. As seen in the architecture diagram [3.2] this class has a dependency in the *SearchStrategy* class which was not implemented at that time. To apply TDD methodology, it was necessary to create a stub. As a stub, the only purpose of *SearchStrategy_Stub* was implementing a dummy behaviour in order to test the Solver class.
The next implemented classes were *SearchStrategy*, *BinarySearchStrategy* and *LinearSearchStrategy* hierarchy.

SearchStrategy is an abstract class which means it cannot be instantiated. Its purpose is to be a "contract" between search strategies, such as linear search and binary search, and classes which use them. Because it has no code, it is not necessary to test it.
However, for its children it is mandatory. The first child, *LinearSearch* implements the linear search algorithm which was in the old PBMin class. The method *init()* encodes the PBConstraints of the problem into a CNF. This encoding is done at the *init()* method because it has to be done only once.
The method *loop()* is the one which implements the algorithm. It starts defining the boundaries of the search with the maximum possible value for the cost function and its minimum.
Because of the values given by the algorithm only differ in one unit, starting on the maximum and ending on the minimum, the functionality "Incremental constraints" from PBLib could be used.
Finally, the method *end()* does nothing.
The other child, *BinarySearchSolver*, implements the binary search algorithm. As before, the method *init()* encodes the problem's constraints into a CNF. The *loop()* method implements the binary search algorithm taking as the left value the minimum possible value of the cost function and as the right value its maximum. Unlike before, the incremental constraints functionality could not be used because the search is not linear. As before, the method *end()* does nothing.

Finally, the last classes to be implemented were *SimpleTimeoutSolver* and *GeneralTimeoutSolver*.
The Template pattern [4] could be applied because the children of *Solver* are a specialization of it. In other words, all the hierarchy shares a lot of behaviour.
The *SimpleTimeoutSolver* only redefines the method *solve()* to a creation of a thread which calls the sat solver with all its implications.
The *GeneralTimeoutSolver* only redefines the method *run()* to a creation of a thread which calls the function *loop()* from the *SearchStrategy* class.

3.3.5 Finalization

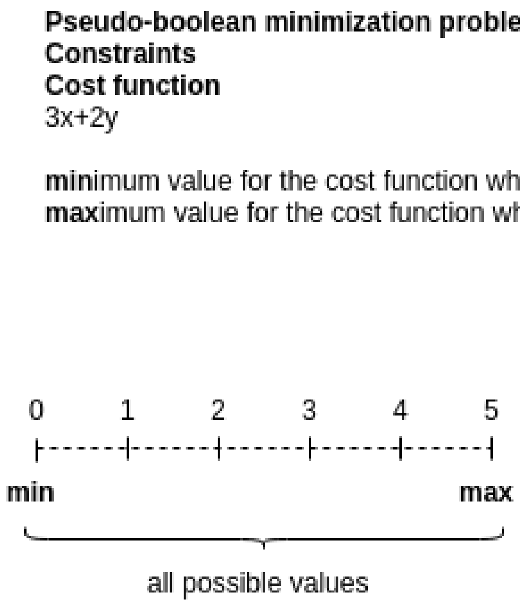
The classes *Solver*, *SimpleTimeoutSolver* and *GeneralTimeoutSolver*, apart from their unit tests required by TDD, were tested with integration tests.
Integration testing was done to expose defects in the interaction between classes and also test the *Solver* hierarchy with real implementations of *SearchStrategy* and see that the timeout was working as expected.

Chapter 4

Search Algorithms

In this chapter, the search algorithms used for this project are explained in more detail and how they have been adapted for this particular application.

These two search algorithms are used to find a target value in a sorted list of values. For this project, what we are trying to find is the minimum value for the cost function, and the list of values are all the integers between the minimum and the maximum possible values for the cost function.



Pseudo-boolean minimization problem
Constraints
Cost function
3x+2y

minimum value for the cost function when {x=false, y = false} = 0
maximum value for the cost function when {x=true, y = true} = 5

FIGURE 4.1: Search Space

As the reader can see from the figure above, the search problem we are facing can be described as one where the search space is formed by integers between *min* and *max* and our target value is unknown.

How can we know that a value is the minimum, i.e. our target?
If m is the minimum value, we know that the constraint formed by $cost\ function \leq m$ is satisfiable whereas the constraint $cost\ function \leq m - 1$ is unsatisfiable.

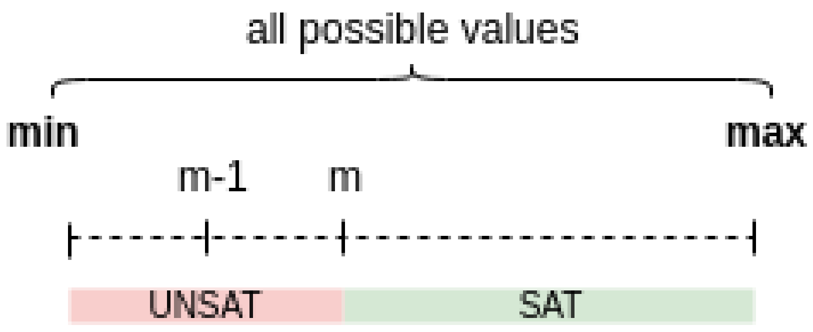


FIGURE 4.2: Satisfiability of the search space given a minimum value

This is the property used to check if the value found is minimum or not.

4.1 Linear search

Linear search is an algorithm which sequentially checks all the values for the target value until it is found or all the elements have been visited.
The search can start with the smallest value or the biggest one. For this application, the initial value is *max* and the algorithm traverse the search space descending until *min*.
This decision was made because proving satisfiability of a problem is usually easier (faster) than proving it is unsatisfiable. Also, given m and n where $n < m$, if $cost\ function \leq m$ is unsatisfiable then $cost\ function \leq n$ will also be unsatisfiable.
As previously said, the algorithm starts with *max*, which means that it generates the Pseudo-Boolean constraint $cost\ function \leq max$. If the problem is unsatisfiable with this value, it can be deduced that with other values it will also be unsatisfiable and therefore the search can end.
Otherwise, it keeps this value as the possible minimum and tries the next one until it finds an unsatisfiable one or the *min*, which is the last value to be checked.
Once the algorithm finished, the last possible value found becomes the minimum, or if no possible minimum was found, the problem is unsatisfiable.

The class *LinearSearchStrategy*[1] implements this algorithm. Let's explain its essential parts:

Firstly, the following variables are initialised:

```
int64_t min_costFunc = pm.getCostFunctionMin();
int64_t k = pm.getCostFunctionMax();
min = pm.getCostFunctionMax()+1;
```

- *min_costFunc* is used to define the last value to look at
- *k* is the value it checks. As previously said, initially is the maximum possible value for the cost function
- *min* is the last minimum value found. Initialised with an invalid value which is the maximum plus one

Then the first constraint is created. The algorithm takes advantage of the *Incremental Constraints* functionality of PBLib.

[1]Source code in Github

```
IncPBConstraint inc_costFunction = IncPBConstraint(w_costFunction, PBLib::LEQ, k);
pb2cnf.encodeIncInital(inc_costFunction, cdb, auxVarManager);
```

It creates an incremental constraint with the $cost\ function \leq k$ and then encodes it into a CNF.

The next part is the loop, the linear search implementation:

```
bool end = false;
while (!end) {
    if (k == min_costFunc) {
        end = true;
    }

    inc_costFunction.encodeNewLeq(k, cdb, auxVarManager);

    bool t_sat;
    (solver->*solve)(temp_model, cdb.getClauses(),t_sat);

    if (t_sat) {
        model = temp_model;
        min = k;
        k--;
    }
    else{
        end = true;
    }
}
```

The loop can end for two reasons:

- *k == min_costFunc*: this is the last value to check
- *t_sat is false*: the current iteration is unsatisfiable, i.e. the next values in the search space will also be unsatisfiable

After encoding the new constraint $cost\ function \leq k$, it calls the solver.

If it is *satisfiable*, the model is stored and *k* is stored as the last *min* found.
If after ending the search no other minimum has been found, that is returned along with its model. After that, a new iteration is done with $k = k - 1$.

4.2 Binary search

Binary search is an algorithm which looks for a value in a given sorted input.
As before, our input is the integers between *min* and *max*, and therefore they are sorted.
In the beginning, Binary search takes the middle value and compares it with the target value. If it is bigger than the target, then it looks for the value in the subspace between minimum value and the value before the middle one. If it is smaller than the target, then it looks for the value in the subspace between the value after the middle one and the maximum.
For this application, the leftmost value of the search is *min*, and the rightmost value

max. Again, our target value is the minimum one. For each iteration, the algorithm checks that the left side of the problem is less or equal than the right side because otherwise, the state of the search would be incorrect.
If this is true, then it gets the middle value between them and generates the constraint $cost\ function \leq middle\ value$, which is added to the other constraints and then encoded into a CNF. If the CNF is satisfiable, then the middle value is stored as a possible minimum, and the search continues with a new space contained between the left side and the integer between the middle value. Otherwise, if the CNF is unsatisfiable, it checks if the stored possible minimum is exactly the integer after the middle value because then the integer would be the minimum value.
If not, it continues the search with a new space contained between the integer after the middle value and the right side.

The class *BinarySearchStrategy*[2] implements this algorithm:

As previously said, the search space is limited between *min* and *max*. For this reason, the variables *left* and *right* are initialised as following:

```
left = p.getCostFunctionMin();
right = p.getCostFunctionMax();
```

As in *LinearSearchStrategy*, the value *min* is initialised as the maximum value for the cost function plus one.

```
min = right + 1;
```

Let's now explain the loop which implements the binary search algorithm:

```
while (not end) {
    if(left<=right){
        cnf.clear();
        cnf.insert(cnf.end(),cnf_constraints.begin(),cnf_constraints.end());

        int64_t k = (left+right)/2;

        if (left == right) {
            //no more values to try
            end = true;
        }

        firstFreshVariable = pb2cnf.encodeLeq(pm.getCostFunction().getWeights(), pm.getCo

        bool t_sat;
        (solver->*solve)(temp_model, cnf,t_sat);

        if (t_sat) {
            min = k;
            right = k-1;
            model = temp_model;
        }
        else{
```

[2]Source code in Github

```
            if (min == k + 1) {
                end = true;
            }
            left = k+1;
        }
    }
    else{
        end = true;
    }
}
```

The search can end for three reasons:

- *left==right*: it is checking the last value, therefore there are no values left for the next iterations
- *k unsatisfiable but k+1 satisfiable*: then *k+1* is the minimum and the search finishes
- *left > right*: the state of the search is inconsistent because it has checked all the values of the space

The first thing it does is clean the CNF and inserting the constraints of the problem again. Then, it calculates the next value to check, which is the one in the middle, and generates the constraint $cost\ function \leq k$. Finally, the CNF is given to the solver.

If the CNF is *satisfiable*, then *k* is stored as the last minimum seen along with its model. The right side of the space is limited to *k-1*.
Otherwise, the left side of the space is limited to *k+1*, and as previously said, it checks if the last minimum value found equals *k+1*.

Chapter 5

Timeouts

In this chapter, the two timeout strategies developed for this project are explained.

As previously said in this document, some users may need a result before a specific time. When timeouts are used, the solution to the problem may not be the optimal one, as explained in the following lines.

5.1 Simple timeout

For this strategy, the user defines a maximum number of seconds that each call to the sat-solver can take. In other words, the amount of time that a value between *min* and *max* is checked to be the optimal or not.

To accomplish this behaviour, a new thread is created, and the call to the sat-solver is executed in parallel while the main thread looks that the elapsed time is less than the timeout defined by the user.

When the elapsed time is longer than the time defined by the user, the main thread kills the sat-solver execution and assumes that the problem was *unsatisfiable*. This assumption leads to suboptimal solutions but correct solutions. However, if the assumption were that the problem was *satisfiable*, then this could point to wrong solutions.

For example:

5.1.1 Linear search

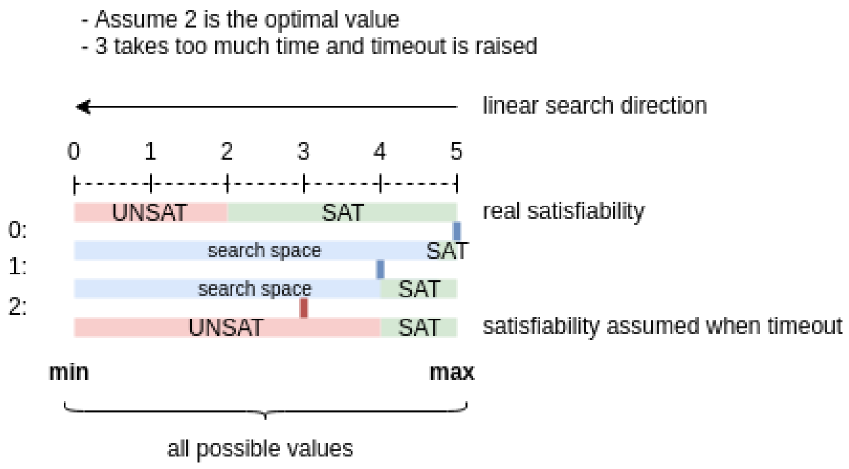


FIGURE 5.1: Simple timeout with linear search

As the reader can see in the figure above [5.1], there is a Pseudo-Boolean minimisation problem which its minimum value is 2, and the cost function values go from 0 to 5. Because it is using *Linear search* algorithm, it will start from 5 and will descend until the problem is *unsatisfiable* or 0.
If 2 is the minimum value, it means that below 2 all is *unsatisfiable* and from all values greater equal than 2 are *satisfiable*. We also assume that the call to the sat-solver when trying 3 will take longer than the time defined by the user, i.e. a timeout will be raised.

At the first iteration of the algorithm, *0:*, the CNF Pseudo-Boolean constraints AND $cost\ function \leq 5$ will be generated which will return SAT.
At the second iteration, *1:*, the CNF Pseudo-Boolean constraints AND $cost\ function \leq 4$ will be generated, and this also will return SAT.
At the third iteration, *2:*, the CNF Pseudo-Boolean constraints AND $cost\ function \leq 3$ will be generated. We know that this should return satisfiable but because it takes too long, the timeout is raised, and unsatisfiable is returned. The algorithm assumes that all the values below 3 will also be unsatisfiable and ends the search. 4 was the last satisfiable value found, so it is returned as the minimum value.

As the reader can see, this is not the optimal solution but a correct one.

5.1.2 Binary search

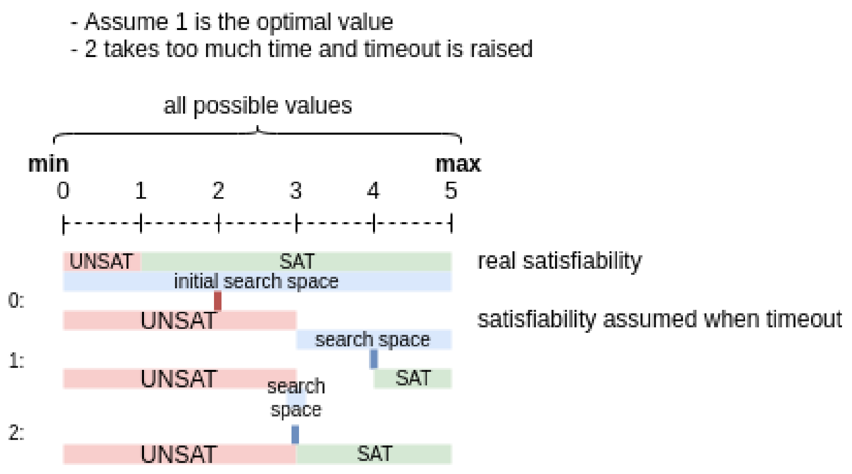


FIGURE 5.2: Simple timeout with binary search

In this case, the search algorithm used is *Binary search*. We assume a Pseudo-Boolean minimisation problem where the minimum value is 1, the values of the cost function are between 0 ad 5, and that the call to the sat-solver when trying 2 will take longer than the time defined by the user, i.e. a timeout will be raised.

At the first iteration, *0:*, the search space is between 0 and 5, and the algorithm knows nothing about the satisfiability of the problem. 2 is the middle value so the CNF Pseudo-Boolean constraints AND $cost\ function \leq 2$ is generated. Because of the problem definition, this is satisfiable, but it takes longer than the time defined by

the user and the timeout is raised.
The execution of the sat-solver is stopped, and it assumes the CNF is unsatisfiable.

At this point, it assumes that 2 and all the values under it are unsatisfiable and the search space is reduced to 3 to 5.
Iterations *1:*, *2:* proceed as a regular Binary search execution, and finally, the value 3 is returned as the optimal one.

As before, the given solution is not optimal but correct.

5.2 General timeout

For this strategy, the user defines a maximum number of seconds that the whole problem can take. In other words, the amount of time that the user will wait for an answer.

To accomplish this behaviour, a new thread is created, and the call to the search algorithm is executed in parallel while the main thread looks that the elapsed time is less than the timeout defined by the user.

When the elapsed time is longer than the time defined by the user, the main thread kills the search algorithm execution and assumes that it finished normally, i.e. the last value found is returned as the minimum one.

5.2.1 Linear search

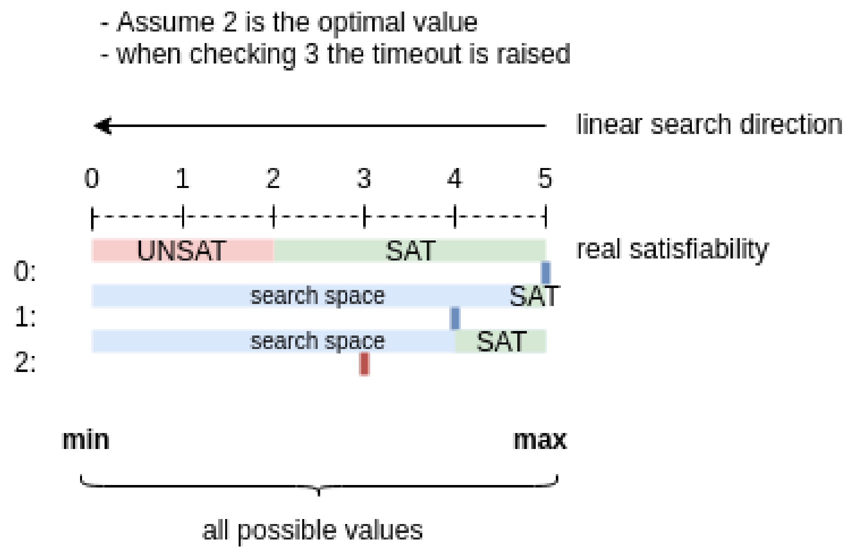


FIGURE 5.3: General timeout with linear search

As the reader can see in the figure above [5.3], there is a Pseudo-Boolean minimisation problem which its minimum value is 2, and the cost function values go from 0 to 5. Because it is using Linear search algorithm, it will start from 5 and will descend until the problem is unsatisfiable or 0.
If 2 is the minimum value, it means that below 2 all is unsatisfiable and from all values greater equal than 2 are satisfiable. We also assume that trying 3 will take longer than the time defined by the user, i.e. a timeout will be raised.

At the first iteration of the algorithm, *0:*, the CNF Pseudo-Boolean constraints AND $cost\ function \leq 5$ will be generated which will return SAT.
At the second iteration, *1:*, the CNF Pseudo-Boolean constraints AND $cost\ function \leq 4$ will be generated, and this also will return SAT.
At the third iteration, *2:*, the CNF Pseudo-Boolean constraints AND $cost\ function \leq 3$ will be generated. Then the timeout is raised, and the last satisfiable value found, 4, is returned as the minimum one.

As the reader can see, this is not the optimal solution but a correct one.

5.2.2 Binary search

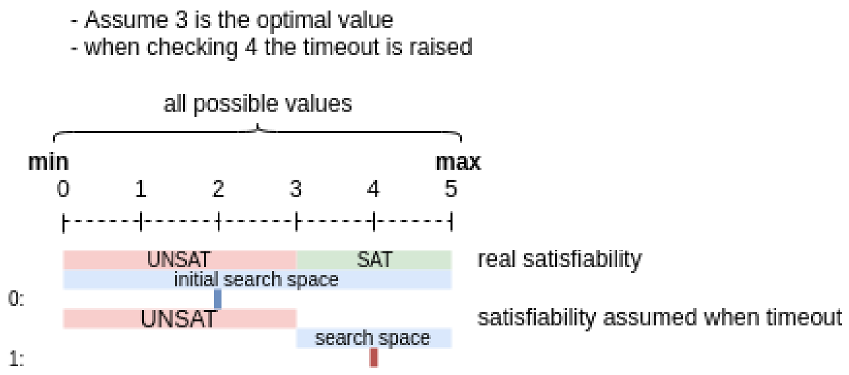

- Assume 3 is the optimal value
- when checking 4 the timeout is raised
all possible values
min
max
0
1
2
3
4
5
UNSAT
SAT
real satisfiability
initial search space
0:
UNSAT
satisfiability assumed when timeout
search space
1:

FIGURE 5.4: General timeout with binary search

In this case, the search algorithm used is Binary search. We assume a Pseudo-Boolean minimisation problem where the minimum value is 3, the values of the cost function are between 0 and 5, and that when trying 4 the timeout will be raised.

At the first iteration, *0:*, the search space is between 0 and 5, and the algorithm knows nothing about the satisfiability of the problem. 2 is the middle value so the CNF Pseudo-Boolean constraints AND $cost\ function \leq 2$ is generated. Because 2 is unsatisfiable, the search space is reduced to 3 to 5, and the next value to be checked is 4.

When checking 4, the timeout is raised, and the search space becomes empty. Because there are no more elements to be checked, the last minimum value found is returned, which in this case there is no one, i.e. the software returns unsatisfiable.

Chapter 6

Project Planning

6.1 Schedule

6.1.1 Estimated project duration

For this project there have been estimated 450 hours of work, starting on **19th of February** and ending on **23rd of June**.

6.1.2 Considerations

The original plan could be modified to be adapted to deviations. Agile methodology implies that some new requirements can appear which could modify the planning. It is hard to do a realistic planning with Agile methodology because the iteration's requirements are not fully known until the Planning stage.

Because this project will be developed sequentially by only one person, the creation of a PERT diagram has been discarded. Nevertheless, some part of the documentation will be done in parallel.

6.2 Resources

For the development of this project, three types of resources will be needed.

6.2.1 Human Resources

- One person working 20 hours per week until the finalization of the project.

6.2.2 Material Resources

- Lenovo IdeaPad U330T
  This laptop will be used to write the documentation and develop the project.

6.2.3 Software Resources

- Trello: Web application to manage project tasks.
- teXstudio: LateX editor to write all the documentation.
- e-mail: Communication tool used to contact the supervisor.
- Atom: Text editor to write the code.

- Git: VCS to backup and keep tracking of the project.
- C++: Language used for the development.
- PBLib: C++ library for Pseudo-Boolean encodings.
- CLion: Code editor focused on C++.
- Google Test: Unit testing framework for C++ developed by Google.

6.3 Project Planning

GEP

This task corresponds to the work done during the GEP course. This task has not any dependency but the work done will be used for the final documentation.

The estimated time for this stage is 70 hours.

Requirements analysis, architecture and debugging

This stage will be used for defining the requisites to accomplish, the architecture of the software and refactor the previous code. Also, the required tools will be installed.

The estimated time for this stage is 90 hours.

Iterations

Because Agile methodology will be followed, the project has been divided into iterations. There will be a total of 3 iterations: Pseudo-Boolean minimisation, Timeout strategies, and Multithreading being this last one optional.

For each iteration, 80h of work are estimated.

Planning
This stage will be used for defining the scope of the iteration and goals.

This stage will be 10 hours long.

Development and TDD
In this stage, the iteration will be developed and tested.

This stage will be 60 hours long.

Finalization
In this stage, all possible bugs will be solved, and feedback from the supervisor will be taken.

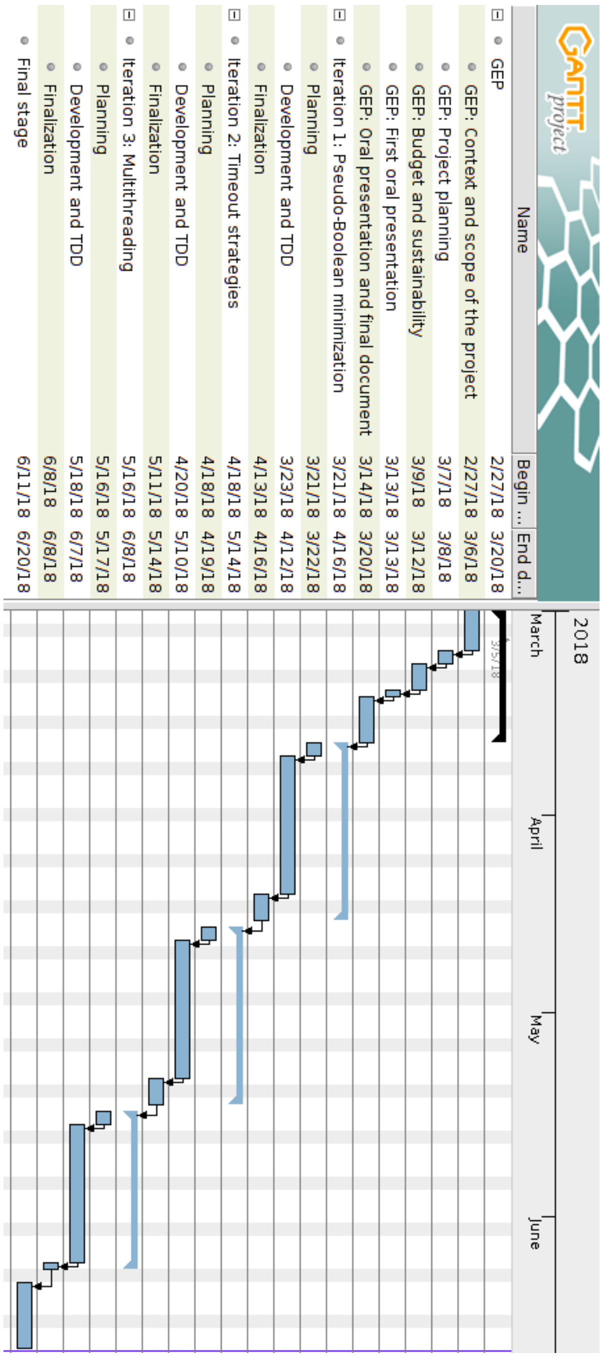
This stage will be 10 hours long.

Final Stage

Here, all the development will be finished, and it will be used for finishing all the documentation and prepare the final presentation.

This stage will take 50 hours.

6.3.1 Gantt Diagram



Name	Begin ...	End d...
GEP	2/27/18	3/20/18
GEP: Context and scope of the project	2/27/18	3/6/18
GEP: Project planning	3/7/18	3/8/18
GEP: Budget and sustainability	3/9/18	3/12/18
GEP: First oral presentation	3/13/18	3/13/18
GEP: Oral presentation and final document	3/14/18	3/20/18
Iteration 1: Pseudo-Boolean minimization	3/21/18	4/16/18
Planning	3/21/18	3/22/18
Development and TDD	3/23/18	4/12/18
Finalization	4/13/18	4/16/18
Iteration 2: Timeout strategies	4/18/18	5/14/18
Planning	4/18/18	4/19/18
Development and TDD	4/20/18	5/10/18
Finalization	5/11/18	5/14/18
Iteration 3: Multithreading	5/16/18	6/8/18
Planning	5/16/18	5/17/18
Development and TDD	5/18/18	6/7/18
Finalization	6/8/18	6/8/18
Final stage	6/11/18	6/20/18

FIGURE 6.1: Gantt diagram of the project

6.4 Alternatives and Action Plans

Because of using an Agile methodology, the project functionalities can be easily adapted during the development.

6.4.1 Potential deviations

Incorrect estimations

It could be that the purposed estimations are not correct and be underestimated or overestimated. In the first case, if some iteration takes less time than expected, the next iteration will be started immediately. On the other case, if an iteration takes more time than expected, more hours will be added, for example, weekend hours.
If even with this countermeasures the iteration is delaying the project some optional improvements will be discarded in order to guarantee the main functionalities and the project would be finished before the deadline. The effect on the resources would be more electricity consumed by the laptop on the added hours.

Summer internship

The deadline for this project is the 23rd of June. The author applied for a CERN internship from 4th of June until 31st of August.
In case of being accepted into the program, the final stage would have to be done during the internship. Since the final stage is for doing the final document and the final presentation, it can be lightened by paralleling part of its work during other stages.
The duration of the project is not affected because the only thing that would be done is a different distribution of the work. For the same reason, it would not have any effect on the resources.

Compatibility issues

As explained before, PBLib could cause some compatibility issues with the existing software which works with another library, Cudd.
If this happens, in the best case some hours will be spent fixing this issue or finding a substitute. In the worst situation, the functionalities of PBLib would have to be implemented which would take much time. The project duration could be affected, but it would be always be finished before the deadline using weekend hours. The resources which will be used in addition is more electricity consumed by the laptop.

Base project code quality

The existing project could have some quality issues and be a source of bugs which would cause many delays. For these reasons, the first stage of this project will be focused on assuring quality code of the existing project. The project duration would not be affected because part of the first stage is focused on that guaranteeing project finalization.

As explained in each possible deviation, the priority is finishing the project before the deadline. As said, extra-hours from the weekend can be used to work on the project to accomplish all the objectives. If even these hours are not enough, then the optional objective *Multithreading* will not be done.

6.5 Project execution

In this section, an analysis of the project execution is done.

Stage	Expected hours	Real hours
GEP	70	70
Requirement analysis, architecture and debugging	90	75
Iteration 1: PB Minimisation	80	87
Iteration 2: Timeout	80	88
Iteration 3: Multithreading	80	-
Finalization	50	65
Total	450	385

TABLE 6.1: Planned time and real time per stage

The first deviation is at *Requirement analysis, architecture and debugging*. The expected hours were 90 but the required were 75. This is because when applying testing methods to the code the amount of errors found was less than expected.

Iteration 1: Pseudo-Boolean minimisation and *Iteration 2: Timeout* took more time than expected specially the second one because of the refactor of the architecture.

In the original plan, after the second iteration came *Iteration 3: Multi-threading*. As already said, the third iteration was optional so instead of it, the author started the final stage. The decision was made knowing that during June he would be doing a full-time internship which would reduce the available time to work on the project significantly.

Chapter 7

Economic Management

In this section, all the costs of the project are exposed.

7.1 Direct costs

Direct costs are those that have a direct relationship with the manufacture of the product. In this case, the only direct costs are the human resources.

7.1.1 Human resources

The cost of the human resources has been estimated with the following expression: $Cost = \frac{Salary}{Hour} \times Expected\ Hours$. The salaries have been extracted from PagePersonnel study [10]. In this study, the salaries are expressed per year. In average, there are 1.500 working hours per year. To obtain the price per hour, the salary per year has been divided by the working hours per year.

Taking into consideration the Gantt chart from the previous deliverable, the dedication of each role has been defined as follows:

Stage	Project Manager	Software Architect	Developer
GEP	70	0	0
Initial Stage	30	30	30
Iteration 1,2,3	6	147	87
Final Stage	30	0	20

TABLE 7.1: Hours destined to each stage per role

Role	Estimated hours (h)	Price/hour (€)	Total cost (€)
Project Manager	136	27	3.672
Software Architect	177	25	4.425
Developer	137	14	1.918
Total			10.015

TABLE 7.2: Human resources budget

7.2 Indirect costs

Indirect costs are those that do not have a direct relationship with the manufacture of the product. In this case, the indirect costs are Hardware, Software, and some others.

7.2.1 Hardware

According to *Agencia Tributaria*[1], the maximum number of years to amortize a computer equipment is 8. Therefore the amortization of Hardware resources has been calculated following this expression: $Amortization = \frac{Price}{8 \times 12} \times 5$

Product	Price (€)	Units	Useful life (y)	Amortization (€)
Lenovo IdeaPad U330T	899	1	8	46,83
Total				46,83

TABLE 7.3: Hardware resources budget

7.2.2 Software

For software resources, free tools have been selected, and student discounts have been used to minimise the total cost.

Product	Price (€)	Units	Useful life (y)	Amortization (€)
GitHub	6,10/month	5	N/A	30,5
GitHub student pack	-6,10/month	5	N/A	-30,5
Clion	6,90/month	5	N/A	34,5
JetBrains Product Pack for Students	-6,90/month	5	N/A	-34,5
Atom	0,00	1	N/A	0,00
TeXstudio	0,00	1	N/A	0,00
Total				0,00

TABLE 7.4: Software resources budget

7.2.3 Other resources

Internet connexion price has been extracted from Pepephone[2] plan, which is 34,6€ per month.
kWh price has been extracted from Selectra. The average price per kWh is 0,12€. In office supplies paper packs, books, pens, ... are included.

7.3 Contingency

The contingency percentage for direct costs has been estimated following the author's experience on past projects. For indirect costs, the budget is more straightforward to estimate therefore a small percentage has been selected.

[1] Agencia Tributaria - amortizations
[2] Pepephone fibra

Product	Price(€)	Units	Total (€)
Internet connexion	0,047/h	450 hours	21,15
Power consumption	51Wh	450 hours	2,75
Print	0,05/page	400 pages	20
Office supplies	50	1	50
Total			93,9

TABLE 7.5: Other resources budget

Concept	Price (€)	Percentage (%)	Total (€)
Direct costs	10.015	30	3.004,5
Indirect costs	140,73	15	21,11
Total			3.025,61

TABLE 7.6: Contingency budget

7.4 Unforeseen

The first unforeseen is that the computer breaks. In this case, a new one will be bought. The other unforeseen events are that the stages of the project being extended. For each stage, a 50% delay has been estimated.

Unforeseen	Cost (€)	Probability (%)	Total (€)
Broken computer	1.300	5	65
Delay GEP stage	945	15	141,75
Delay initial stage	990	15	148,5
Delay iteration 1	842,5	15	126,38
Delay iteration 2	842,5	15	126,38
Delay iteration 3	842,5	15	126,38
Delay final stage	545	15	81,75
Total			816,14

TABLE 7.7: Unforeseen budget

7.5 Total budget

In conclusion, the total budget of the project is:

	Cost (€)
Direct costs	10.015
Indirect costs	140,73
Contingency	3.025,61
Unforeseen	816,14
Total	13.997,48

TABLE 7.8: Total budget

7.6 Control management

The control management mechanisms will be used to study and compare deviations.

The Human Resources is an initial estimation, therefore, the estimated cost and the real cost obtained once the project is finished will be compared. In any case, an hour follow-up will be done for each iteration and the functionalities implemented to see if the planning is accurate, or correct possible deviations and decide which functionalities could be added or deleted in order to accomplish the planning. Another method to solve the possible deviations could be reorganizing the Gantt chart.

At the end of the project, the original estimated budget will be compared with the real one. Finally, a study of which deviations and unforeseen appeared will be done and check if they can be covered by the contingency budget. This analysis will be very useful to do future budgets and to apply the extracted conclusions.

The indicators used for that are: Variance in cost by rate, efficiency variance, variance in totals, . . .

7.7 Real budget

In this section, an analysis of the deviation on the budget is done.

Direct costs

Stage	Product Manager (h)	Software Architect (h)	Developer (h)
GEP	70	0	0
Requirement analysis, architecture and debuggin	30	30	15
Iteration 1: Pseudo-Boolean minimisation	2	49	36
Iteration 2: Timeout	2	49	37
Final Stage	60	0	5

TABLE 7.9: Hours per role

In this table [7.9], the hours of each iteration are divided per role.
As the reader can see, the direct costs have been smaller than expected. As ex-

Role	Price/Hour (€/h)	Hours (h)	Cost (€)
Product Manager	27,00	164	4.428,00
Software Architect	25,00	128	3.200,00
Developer	14,00	93	1.302,00
Total			8.930,00

TABLE 7.10: Real cost of human resources

plained in the previous chapter [6], the Iteration 3: Multi-threading has not been done therefore the number of hours for each role has also decreased.

Indirect costs

There has been two additions to the Indirect Costs for the project which is a new software:

- Grammarly[3]: The Grammarly Premium feature was bought in order to improve the English level for the document.
- Transport: In order to do the final presentation, a plant ticket from Geneva to Barcelona was required.

Product	Price(€)	Units	Total (€)
Grammarly Premium	139,95	1	139,95
Grammarly discount	-74,95	1	-74,95
Transport	112	1	112
Total			177,00

TABLE 7.11: Other resources budget

Contingency and unforeseen

The budget for these two cases has not been necessary and therefore they remain unchanged.

Total cost

In conclusion, the total budget of the project is: The difference between the estimated

	Cost (€)
Direct costs	8.930,00
Indirect costs	257,73
Contingency	3.025,61
Unforeseen	816,14
Total	13.029,48

TABLE 7.12: Total cost

cost, 13.977,48€, and the real one, 13.029,48€, is 948,00€.

As previously mentioned, the difference is due to the reduction of hours needed for the project.

However, the final price to the costumer would be 9.187,73€ because Contingency and Unforeseen budget has not been required.

[3]https://www.grammarly.com

Chapter 8

Sustainability and Social Commitment

8.1 Sustainability Matrix

In this section, the sustainability matrix is resumed according to the numbers described here [5].

	PPP	Useful life	Risks
Environmental	7	20	-4
Economical	7	15	-10
Social	8	15	0
Sustainability range	58		

TABLE 8.1: Sustainability matrix

8.2 Economic dimension

8.2.1 PPP

The estimated budget of the project can be found in table [7.8]. The estimated budget is 13.997,48€. This number has been estimated taking into account the working hours of each role, the hardware and software used, indirect costs, contingency, and unforeseen events.

8.2.2 Shelf life

Nowadays, the no optimization of Pseudo-Boolean encodings implies that the problems are bigger and harder which causes a long execution and more consumption of resources. With the optimizations that this project will study, the final execution time could be reduced and therefore the power needed to solve the problem which translates into a more reduced cost.

8.2.3 Risks

As exposed previously, some risks are problems with the planning, problems with the tools used,...
The main risk is that the optimizations proposed are not useful in a practical environment.

8.3 Environmental dimension

8.3.1 PPP

The estimated electric usage for this project can be found in this table [7.5]. The estimation has been done with this expression: $E = \frac{W}{h} \times T$. In this project $E = \frac{51W}{1h} \times 450h = 22,950kW$

It is hard to minimise more the impact of this project. Some strategies are turning off the computer when not using it, minimising the amount of paper used, ...
Some resources are reused, for example, instead of writing all the functionalities, some C++ libraries will be used.

8.3.2 Shelf life

It is hard to measure the footprint of this project along with all its useful life. It will depend on the success of the project, and how many people will use it.

Currently SAT problems are executed in SAT-Solvers using some optimizations. As explained before, this problem is NP-Complete which among other things implies that there is no known algorithm which can solve it in polynomial time. In other words, solving SAT is very time and resource expensive.
Also, SAT is widely used in many fields. For example, computational complexity, databases, programming languages, artificial intelligence and system verification. This translates into a big electricity consumption and a huge footprint. For example, the MareNostrum [7] supercomputer spends 1,3MW/year.

This project purposes more optimizations to reduce the execution time. Even if these optimisations are small, because SAT is widely used, it could have a huge impact. It will have a positive impact because it will reduce the total CO^2 emissions released by the computers used to solve them.

8.3.3 Risks

The footprint of this project could be worst than expected if the development of it is extended.

8.4 Social dimension

8.4.1 PPP

This first stage of the project, GEP, will improve the author's management and planning skills, his English abilities, how to document and budget projects.
The other stages will expand his knowledge about informatics and the opportunity to put in practice a lot of skills developed during this degree.
Finally, his ability to present in front of people and defend the work done during these months.

8.4.2 Shelf life

This project will improve a lot of fields because SAT-Solvers are widely used. For example, Planners, Artificial Intelligence, ... which can have an unpredictable impact in the life of people.

Currently this problem is solved using other techniques. The solution that this project purposes is an addition to them (it is not exclusive). There is a real need for this type of projects because as said previously, SAT is an NP-Complete problem therefore any improvement on this field will reduce the hardness of the problem with all the consequences this implies.

8.4.3 Risks

The only negative impact that this project can have is not being used. In this case, it will not be used and the society will remain unchanged.

Chapter 9

Experiments and results

In this chapter the most relevant experiments done are explained.

All these experiments were done on a laptop with the following characteristics:

- 8 GB of RAM
- Intel® Core™ i7-4500U CPU @ 1.80GHz × 4
- Ubuntu 16.04 LTS

9.1 Pseudo-Boolean Minimisation benchmarks

The goal of this experiment was to compare the time required to solve Pseudo-Boolean minimisation problems with *Linear search* and *Binary search*.
The International Center for Computational Logic[1] has available some benchmarks from their previous competitions, in particular, MINLPLIB2 which offers Pseudo-Boolean optimisation problems.

From that problems, four of them were selected:

- minlplib2-pb-0.1.0/opb/autocorr_bern20-03.opb
  variable= 20 constraint= 1 product= 18 sizeproduct= 36
- minlplib2-pb-0.1.0/opb/hmittelman.opb
  variable= 16 constraint= 7 product= 9 sizeproduct= 44
- minlplib2-pb-0.1.0/opb/sporttournament10.opb
  variable= 45 constraint= 1 product= 80 sizeproduct= 160
- minlplib2-pb-0.1.0/opb/crossdock_15x7.opb
  variable= 210 constraint= 44 product= 2793 sizeproduct= 5586

To convert these benchmarks to a syntax which could be understood by the software, a python script was made.

- The constraints and the cost function were not linear, i.e. of the form $w_1v_1v_2 + w_2v_2v_3 \geq k$. These expressions were simplified keeping only the first variable of each term: $w_1v_1 + w_2v_2 \geq k$
- The relational operator in the constraints was $\geq$. These were converted to equivalent constraints with $\leq$. To do that, the script negates all the weights.

[1]http://pbeva.computational-logic.org/

For each benchmark, two files were created: one using the Linear search algorithm and the other using Binary search algorithm.
For minlplib2-pb-0.1.0/opb/autocorr_bern20-03.opb:

- benchmarks_files/bench_1_bin.cpp
- benchmarks_files/bench_1_lin.cpp

For minlplib2-pb-0.1.0/opb/hmittelman.opb:

- benchmarks_files/bench_2_bin.cpp
- benchmarks_files/bench_2_lin.cpp

For minlplib2-pb-0.1.0/opb/sporttournament10.opb:

- benchmarks_files/bench_3_bin.cpp
- benchmarks_files/bench_3_lin.cpp

For minlplib2-pb-0.1.0/opb/crossdock_15x7.opb:

- benchmarks_files/bench_4_bin.cpp
- benchmarks_files/bench_4_lin.cpp

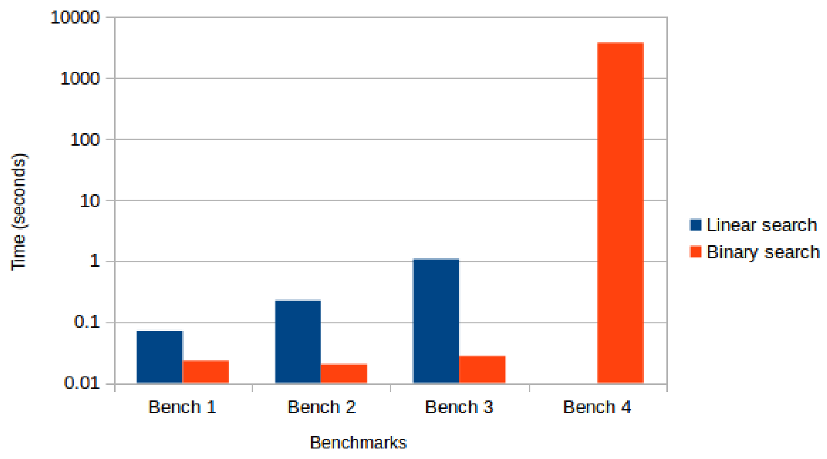


FIGURE 9.1: Time per search algorithm for each benchmark

As the reader can see in the figure above [9.1], in all the benchmarks Binary search was faster than Linear search. In the last one, Binary search took 3.77,62 seconds (1,05 h), whereas Linear search did not end after 8 hours of execution. That is the reason why it is not represented.

However, even that Linear search is slower than Binary search, its cost function encoding into CNF using Incremental Constraints was considerably faster than the conventional encoding used in Binary search.

Chapter 10

Conclusions

This work was focused on a set of topics with the common goal of improving the overall time required to solve Pseudo-Boolean minimisation problems, and to introduce a more user-friendly interface to define these problems.

The first goal was creating a layer between the user and PBLib which hid it and simplified the variable's management. This layer allows the definition of Pseudo-Boolean formulae, Pseudo-Boolean constraints, and Pseudo-Boolean minimisation problems. This layer also allows defining which search strategy algorithm is used to find the minimum value for the cost function. Two algorithms have been implemented, *linear search* and *binary search*. With the first one, the functionality Incremental Constraints could be used in order to optimise the encoding into CNF for each iteration. In both cases, the software takes advantages of PBLib encodings in order to generate the CNF.

The second part of this project was focused on incorporating timeout strategies. This goal was set for users who needed results before a specific deadline and valued more time than the optimal result. The first type, general timeout, stops the execution of the search algorithm and return the last minimum value found. On the other hand, simple timeout, stops the execution of the solver for the selected value and continues the search as if that CNF was unsatisfiable.

Altogether, this project ended in a C++ framework which allows working with Boolean formulae and Pseudo-Boolean formulae.

10.1 Future Work

As previously explained in Project Scope, one of the goals of this project was implementing multi-threading techniques to split the work and solve the problems in less time. Due to time restrictions, this goal could not be done. Although it was optional, the author expects to add this functionality and keep improving the tool.

In order to make this tool usable by other people, it is necessary to create an installer which simplifies all the process. Even though the header files are documented, it would be great to create some documentation with code examples, and more in-depth explanations were done.

Currently, the software only supports Pseudo-Boolean minimisation but not Pseudo-Boolean maximisation even maximisation problems can be easily converted to minimisation and the inverse. Finally, support other relational operators for the constraints, such as greater equal ($\geq$) and equal ($=$), although all the constraints can be converted to less equal.

Appendix A

More information

A.1 Why SAT Solvers use CNF as input format?

There are two mains reasons for this: Equisatisfiability and Computational Complexity. Let us start with the first one:

Two *BF* are **equisatisfiable** if and only if both have the same *models*. This may seem the same as equality but it is not because in an equality relationship both *BF* have to have the same variables.
This is important because between a *BF* and its result from a *CNF* transformation the equisatisfiability is preserved which means that if the *SAT Solver* finds a *model* for the *CNF*, then this *interpretation* will be also a *model* for the original *BF*.

The second reason is computational complexity. Let us have a look at the following table: So as a *BF* can be converted into a *CNF* in linear time while preserving

	DNF	CNF
TAUT	NP	P
SAT	P	NP

TABLE A.1: Complexity of deciding if a *BF* is SAT or TAUT depending of its format.

equisatisfiability, *SAT Solvers* will use them to target satisfiability.

Appendix B

Google Test

In this appendix, Google Test is explained in more detail, along with some examples and how it has been applied to this application.

B.1 Google Test

Google Test, or Google C++ Testing Framework, is a tool developed by Google. As a Testing Framework, it allows the definition of functions called *TEST()* which do two things:

- Executes some functionality of the code
- Compares the output with the expected output to detect unexpected and erroneous behaviour.

Let us evaluate the following test:

```
TEST(FactorialTest, Positive) {
    EXPECT_EQ(1, Factorial(1));
    EXPECT_EQ(2, Factorial(2));
    EXPECT_EQ(6, Factorial(3));
    EXPECT_EQ(40320, Factorial(8));
}
```

FactorialTest is the test case name which is used for defining test groups. For example, if the programmer wanted to test Factorial behaviour for negative input, he/she could have difined *TEST(FactorialTest,Negative)*. Positive is the test name.
As the reader can see, what the test is doing is checking that the output of the function is equal to the expected output. *EXPECT_EQ(1,Factorial(1));*

B.2 Google Test for this project

All the tests done for this project can be found inside the *test_files* folder.
The folder contains three types of files:

- *_UT.cpp : Unit Tests
- *_INT.cpp : Integration Tests
- *_Stup.cpp : Stub class

Unit testing is a software testing technique where individual units/components of the software are tested. Its purpose is to validate that each component works as expected. The majority of tests done for this project are Unit Tests. The first reason is that the used methodology has been Test Driven Development. The second reason is that Unit Tests are more straightforward to generate and they reveal unexpected beahviour closer to the source.
An example of UT in this project could be *PBMin_UT.cpp*:

```
TEST(GetFirstFreshVariable,getFirstFreshVariable){
    std::vector< PBConstraint > e_constraints = {
        PBConstraint(PBFormula({1,2},{1,2}),1),
        PBConstraint(PBFormula({3,4},{2,3}),1),
        PBConstraint(PBFormula({3,7},{1,3}),1)
    };
    PBMin m = PBMin(e_constraints, PBFormula({3,-5},{-1,2}));
    EXPECT_EQ(m.getFirstFreshVariable(), 4);
}
```

For example, this test checks that the implementation of the function *getFirstFresh-Variable()* works as expected.
Integration testing is a software testing where individual units/components of software are combined and tested as a group. This type of testing has been used for testing the solvers with the search strategies which depend on them.
An example of INT in this project could be *GeneralTimeoutSolver_BinarySearchStrategy_INT.cpp*:

```
TEST(Solve,problem3){
    std::vector< PBConstraint > constraints = {
        PBConstraint(PBFormula({2,2},{1,-1}),1),
        PBConstraint(PBFormula({3,4},{2,3}),1),
        PBConstraint(PBFormula({3,7},{1,3}),1)
    };
    PBFormula costFunction({-1,-3,7,-5},{1,-1,2,-2});
    bool e_sat = false;
    int64_t e_min = -8;
    std::vector< int32_t > e_model = {-1};
    BinarySearchStrategy bs;
    PBMin m = PBMin(constraints, costFunction);
    GeneralTimeoutSolver s(5,&bs,m);
    std::vector< int32_t > model;
    int64_t min;
    bool sat = s.run(model,min);
    EXPECT_EQ(sat, e_sat);
}
```

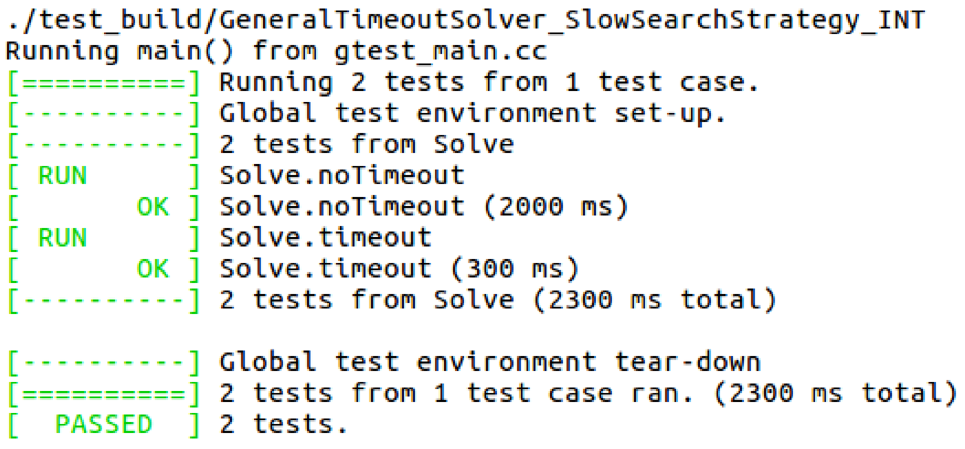
This test how *GeneralTimeoutSolver* and *BinarySearchStrategy* work together.
Finally, a Stub class was done in order to test a component which relies on an unimplemented class at that time. In a nutshell, a Stub class is a class which simulates a behaviour.
An example of a Stub class in this project could be *SlowSearchStrategy_Stub.cpp*:

```
void loop(void (Solver::*solver)(std::vector< int32_t > &, const
std::vector< std::vector< int32_t > > &, bool &),std::vector< int32_t > &
model, int64_t & min, bool &sat, Solver *s, const PBMin &p) override {
    sleep(2);
    model.clear();
    model.push_back(1);
    model.push_back(2);
    model.push_back(3);
    min = 123;
    sat = true;
}
```

This class calls the *sleep(2)* function in order to test the implemented timeouts. As the reader can see, it is not a real implementation of a *Solver* because its purpose is not to test itself but the timeouts.

B.3 Google Test results

Once a test file is compiled, it can be executed. Google Test framework will execute all the tests and do the required checks. The output is shown through the terminal.



```
./test_build/GeneralTimeoutSolver_SlowSearchStrategy_INT
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from Solve
[ RUN      ] Solve.noTimeout
[       OK ] Solve.noTimeout (2000 ms)
[ RUN      ] Solve.timeout
[       OK ] Solve.timeout (300 ms)
[----------] 2 tests from Solve (2300 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (2300 ms total)
[  PASSED  ] 2 tests.
```

FIGURE B.1: Google Test results from command line

The figure above [B.1] was the output from *GeneralTimeoutSolver_SlowSearchStrategy_INT* test suite. As the reader can see, the output from the test suite was the name of the test executed and if it passed the checks or not. If a test does not pass the checks, the output becomes red, and the comparison which failed is printed.

Bibliography

[1] Ignasi Abío et al. "BDDs for Pseudo-Boolean Constraints – Revisited". In: ().

[2] Stephen A. Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. New York, New York, USA: ACM Press, 1971, pp. 151–158. DOI: 10.1145/800157.805047.

[3] Rafael Farré et al. *Notas de Clase para IL - 2.Definición de la Lógica Proposicional*. Barcelona, 2009.

[4] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. ISBN: 0201633612.

[5] Jordi Garcia et al. "Artículo invitado La sostenibilidad en los proyectos de ingeniería". In: (). URL: https://upcommons.upc.edu/bitstream/handle/2117/23240/127-1047-1-PB.pdf.

[6] Steffen Hölldobler, Norbert Manthey, and Peter Steinke. "A Compact Encoding of Pseudo-Boolean Constraints into SAT". In: ().

[7] *MareNostrum 4 supercomputer to be 12 times more powerful than MareNostrum 3 | BSC-CNS*. URL: https://www.bsc.es/news/bsc-news/marenostrum-4-supercomputer-be-12-times-more-powerful-marenostrum-3 (visited on 04/06/2018).

[8] Robert C. Martin. *Design Principles and Design Patterns*.

[9] David Mitchell, Bart Selman, and Hector Levesque. "Hard and Easy Distributions of SAT Problems". In: ().

[10] PagePersonnel. "Selección y trabajo temporal especializado". In: (). URL: https://www.pagepersonnel.es/sites/pagepersonnel.es/files/er{_}tecnologia16.pdf.

[11] Tobias Philipp and Peter Steinke. "PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF". In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Vol. 9340. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 9–16. ISBN: 978-3-319-24317-7. DOI: 10.1007/978-3-319-24318-4_2.

[12] Peter Steinke. "PBLib – A C++ Toolkit for Encoding Pseudo-Boolean Constraints into CNF". In: (2015). URL: http://tools.computational-logic.org/content/pblib/pblib.pdf.