Design of an environment for solving pseudo-Boolean optimization problems

Author: Marc Benedí
Supervisor: Dr Cortadella
CS - FIB - UPC

Index

- Introduction
- Basic concepts
- The base project
- Pseudo-Boolean minimisation
- Bachelor's thesis
- Conclusions and future work
- Analysis of the planning
- Sustainability

Introduction

- C++ library
- Reduce time to solve pseudo-Boolean minimisation problems
- Build onto an existing one, the *Base Project*
- Pseudo-Boolean constraints are encoded with *PBLib*
- Two search algorithms to find the optimal value
- Two timeout strategies

Basic concepts

Boolean formula

- Variables

$$a, b, c, \ldots \in \mathbb{B}$$

- Connectors

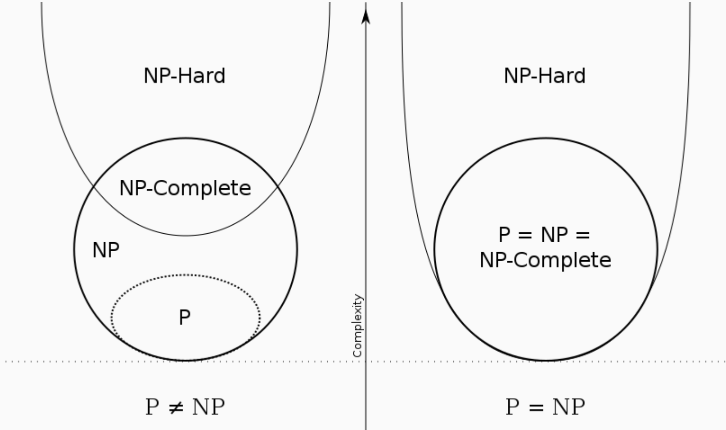
$$\wedge, \vee, \neg$$

- For exemple

$$a \wedge b \vee (\neg c)$$

Boolean satisfiability problem

- Given a propositional formula f, is there a truth assignment i such that $i(f) = 1$?
- NP-complete



https://en.wikipedia.org/wiki/NP-completeness#/media/File:P_np_np-complete_np-hard.svg

Conjunctive Normal Form

- Clause: Finite disjunction of literals

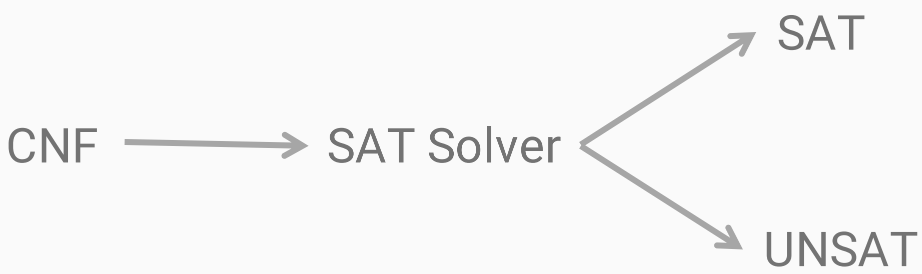
$$l_1 \lor \cdots \lor l_n$$

- CNF: Conjunction of one or more claues

$$(A \lor B) \land C$$

SAT Solver

- Software for solving SAT



Example

- Boolean formula:

$$a \vee (b \wedge c)$$

- CNF:

$$(a \vee b) \wedge (a \vee c)$$

- Solver

$$Satisfiable$$

$$I_1(a) = 1, I_1(b) = 0, I_1(c) = 0$$

Base project

Motivation

- Existing encoding to CNF
- New transformation

- Reduce the number of clauses adding auxiliar variables
- See the tradeoffs

Requirements

- Straightforward notation: literals and operators

```
Formula a = BoolFunc::newLit("a");
Formula b = BoolFunc::newLit("b");

Formula f = (a+b) * b;
```

Transformations

```
Formula f = (a+b)*b;
Cnf cnf = CnfConverter::tseitin(f);
```

- Tseytin
  - Length CNF is linear in the size of the formula
  - Equisatisfiable

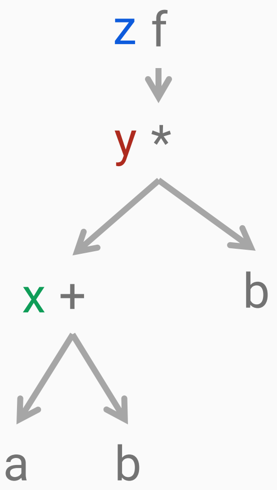
$x = a \vee b$

$a \rightarrow x, b \rightarrow x, x \rightarrow (a \vee b)$

$y = x \wedge b$

$y \rightarrow x, y \rightarrow b, (x \wedge b) \rightarrow y$

z



Transformations

```
Formula f = (a+b)*b;
//Convert formula to bdd
Cnf cnf = CnfConverter::convertToCnf(f_bdd);
```

- Extracting primes

$f = (a * b) + (c * d)$

$cnf = (a + c) * (a + d) * (b + c) * (b + d)$

$\overline{f} = (\overline{a} + \overline{b}) * (\overline{c} + \overline{d})$

$SoP = (\overline{a} * \overline{c}) + (\overline{a} * \overline{d})$

$+ (\overline{b} * \overline{c}) + (\overline{b} * \overline{d})$

- Cudd - Colorado University Decision Diagram package

cd \ ab	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	1	1	1	1
10	0	0	1	0

cd \ ab	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	0	0	0	0
10	1	1	0	1

http://vlsi.colorado.edu/~fabio/

Transformations

- Problem

$$f = XOR(a,b,c,d)$$

$$\begin{aligned} cnf = {} & \overline{abcd} + \overline{ab}cd \\ & + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} \\ & + a\overline{bc}d + a\bar{b}c\bar{d} \\ & + ab\overline{cd} + abcd \end{aligned}$$

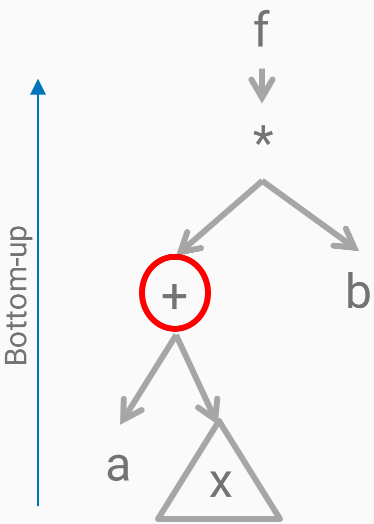
cd \ ab	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

- Idea: When function's primes cover a small space then a lot of clauses are required

Transformations

```
Formula f = (a+b)*b;
MixCNFConverter m = MixCNFConverter();
m.convert(f);
Cnf cnf = m.getResult();
```

- Extracting primes and adding new variables



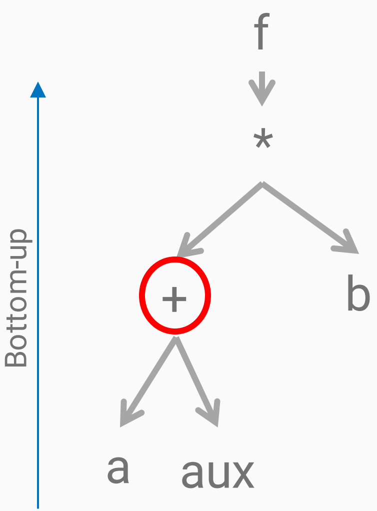
- Build BDD subtree
- Look at the largest cube ratio

- If not big enough
  - Get biggest child (x)
- If big enough generate CNF extracting primes

Transformations

```
Formula f = (a+b)*b;
MixCNFConverter m = MixCNFConverter();
m.convert(f);
Cnf cnf = m.getResult();
```

- Extracting primes and adding new variables

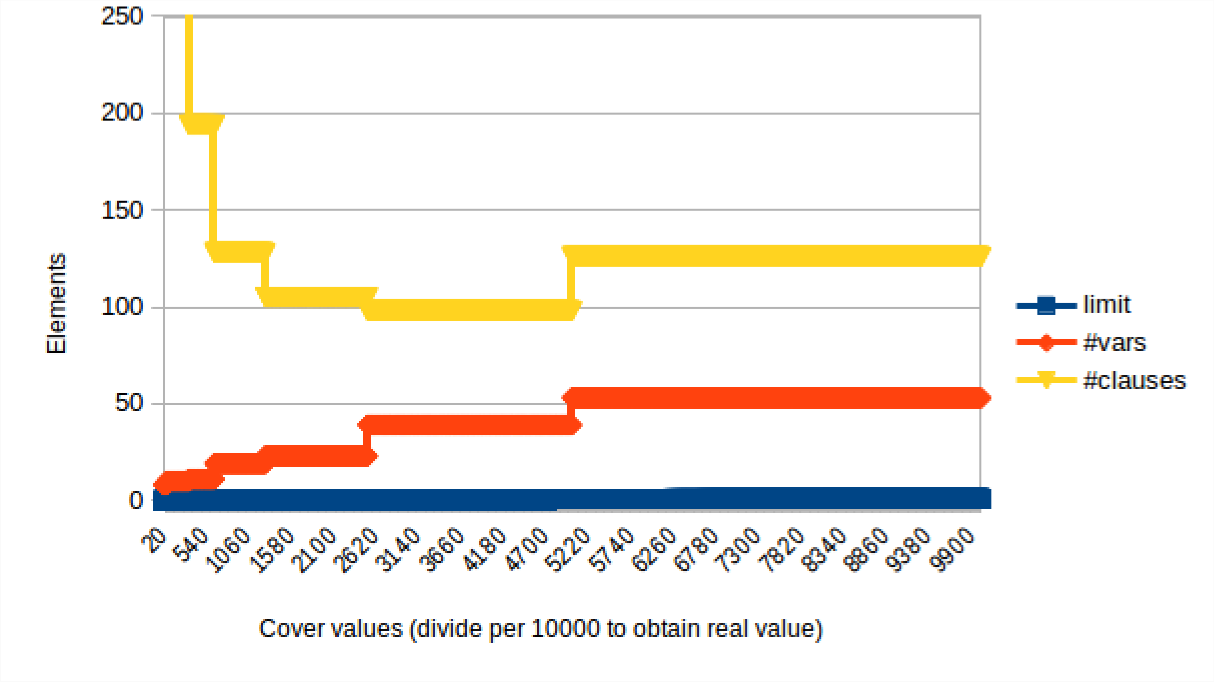


- Build BDD subtree
- Look at the largest cube ratio

- If not big enough
    - Get biggest child (x)
    - Replace it by an auxiliary variable
    - Add x XNOR aux
- If big enough generate CNF extracting primes

- Continue with the parent node

Experiment


250
200
150
100
50
0
Elements
20
540
1060
1580
2100
2620
3140
3660
4180
4700
5220
5740
6260
6780
7300
7820
8340
8860
9380
9900
Cover values (divide per 10000 to obtain real value)
limit
#vars
#clauses

The problem:
Pseudo-Boolean optimisation

Pseudo-Boolean optimisation

- Pseudo-Boolean constraints

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n \# k$$

$$w_i, k \in \mathbb{Z}$$

$$x_i \in \mathbb{B}$$

$$\# \in \{=, \leq, \geq, <, >\}$$

- Cost function

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

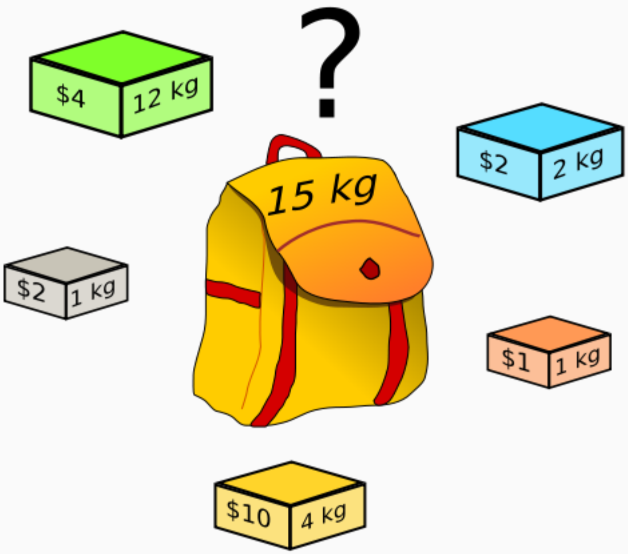
$$w_i \in \mathbb{Z}$$

$$x_i \in \mathbb{B}$$

maximize	$\mathbf{c}^{\mathrm{T}}\mathbf{x}$
subject to	$A\mathbf{x} \leq \mathbf{b}$
and	$\mathbf{x} \geq \mathbf{0}$

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

Example - Knapsack problem



https://upload.wikimedia.org/wikipedia/commons/thumb/f/fd/Knapsack.svg/250px-Knapsack.svg.png

Variables:

$$o_1, o_2, \dots, o_n \in \mathbb{B}$$
$$w_1, w_2, \dots, w_n \in \mathbb{Z}$$
$$v_1, v_2, \dots, v_n \in \mathbb{Z}$$

Constraints:

$$w_1 o_1 + w_2 o_2 + \cdots + w_n o_n \leq knapsack's\ capacity$$

Cost function:

$$v_1 o_1 + v_2 o_2 + \cdots + v_n o_n$$

TFG software

Objectives

- Pseudo-Boolean minimisation
- Timeouts
- Multithreading*

PBLib

- C++ library
- Encodings maintain arc consistency by unit propagation
- Decides which encoder provides the most effective translation
- Variable's management done by the user

At most one	*At most K*	*PB*
sequential*	BDD**	BDD
bimander	cardinality networks	adder networks
commander	adder networks	watchdog
k-product	todo: perfect hashing	sorting networks
binary		binary merge
pairwise		sequential weight counter
nested		
* equivalent to BDD, latter and regular encoding ** equivalent to sequential counter Encodings labeled with *todo* are planed for the (near) future.		

http://tools.computational-logic.org/content/pblib.php

Pseudo-Boolean minimisation layer

- Variables *int32_t*
- Coefficients *int64_t*
- Compact representation

Maximum compatibility with PBLib

```
PBFormula pf = PBFormula({-3,2}, {1,-2});
```

$$-3x_1 + 2\overline{x_2}$$

```
PBConstraint c = PBConstraint(PBFormula({3,2},{1,2}),1);
```

$$3x_1 + 2x_2 \leq 1$$

Pseudo-Boolean minimisation layer

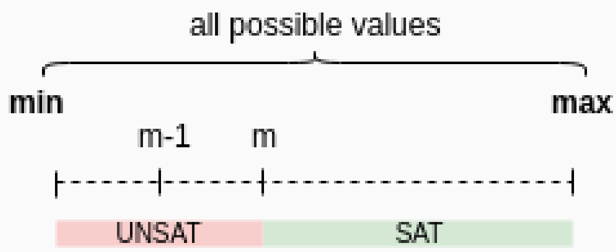
```
std::vector< PBConstraint > constraints = {
    PBConstraint(PBFormula({1,2},{-1,-2}),1),
    PBConstraint(PBFormula({3,4},{2,-3}),1),
    PBConstraint(PBFormula({3,7},{1,-3}),1)
};

PBMin m = PBMin(constraints, PBFormula({3,-5},{4,5}));
```

$$\begin{array}{c} \overline{x_1} - 2\overline{x_2} \leq 1, \\ 3x_2 + 4\overline{x_3} \leq 1, \\ 3x_1 + 7\overline{x_3} \leq 1 \\ \hline 3x_4 - 5x_5 \end{array}$$

Search algorithms

Search space



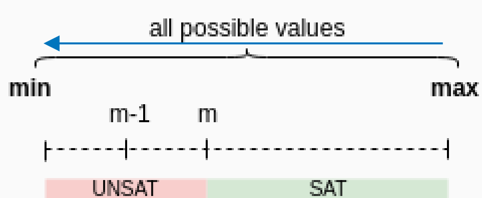
Linear search

```
LinearSearchStrategy ls;
PBMin m = PBMin(constraints, costFunction);
Solver s(&ls,m);
```

Pseudo-Boolean constraints → PBLib → CNF constraints

Cost function, Linear search → PBLib → CNF cost function <=k

SAT solver



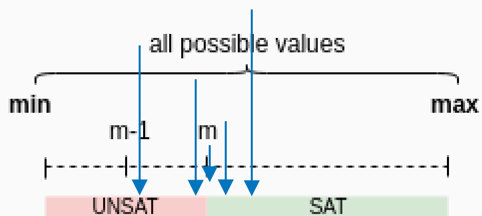
Binary search

```
BinarySearchStrategy bs;
PBMin m = PBMin(constraints, costFunction);
Solver s(&bs,m);
```

Pseudo-Boolean constraints → PBLib → CNF constraints

Cost function, Binary search → PBLib → CNF cost function <=k

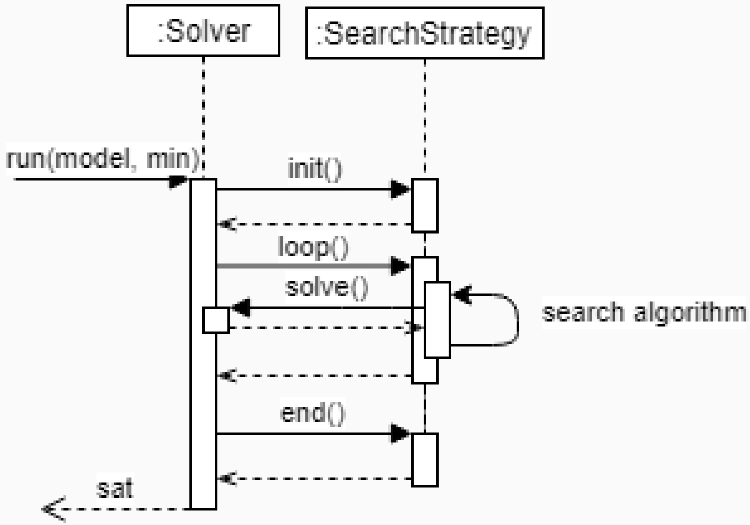
SAT solver



Timeouts

Base solver

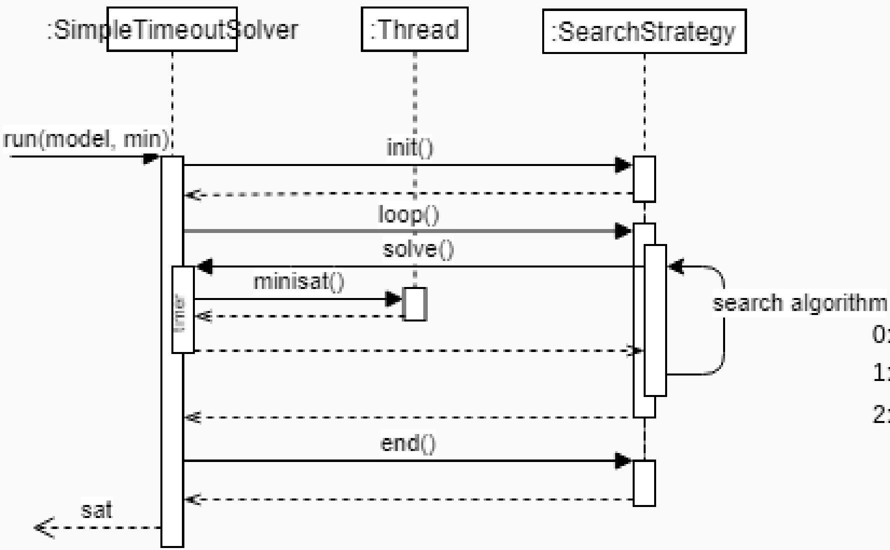
```
Solver solver(&search,problem);
```



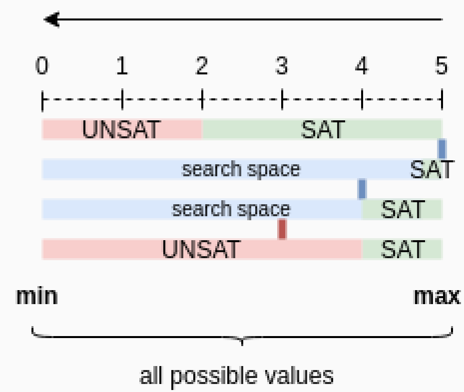
Simple timeout

```
SimpleTimeoutSolver s(timeout,&search,problem);
```

- Timeout for each call to the SAT Solver



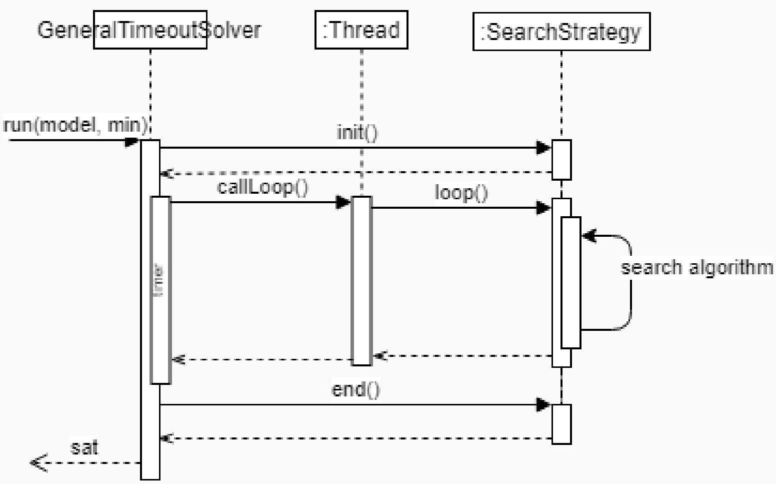
- Assume 2 is the optimal value
- 3 takes too much time and timeout is raised

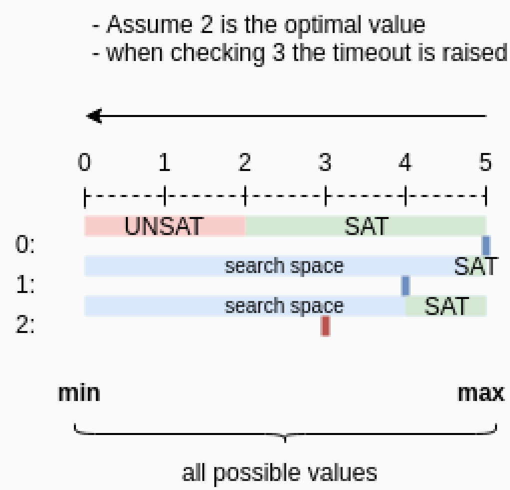


General timeout

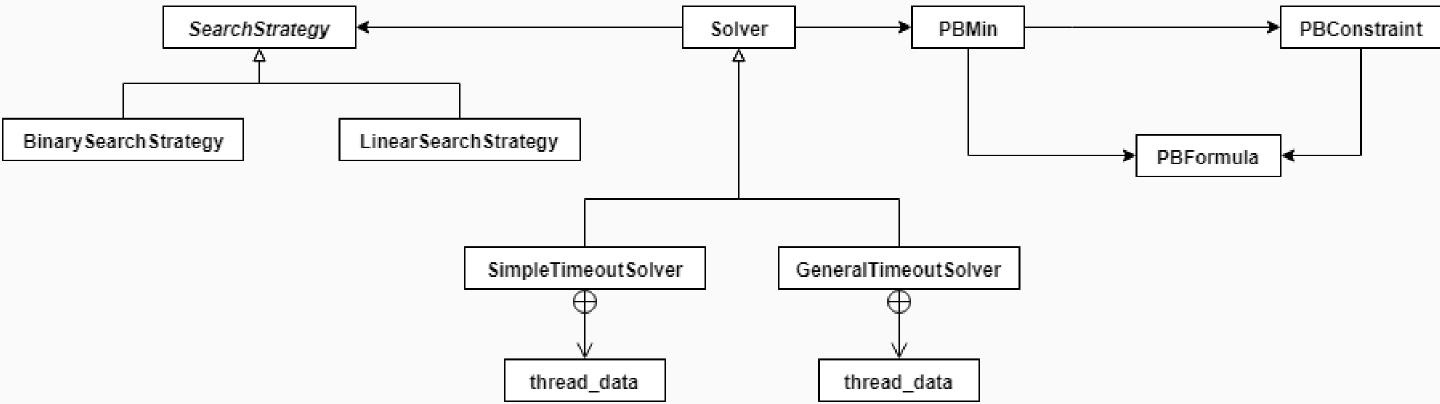
```
GeneralTimeoutSolver s(timeout,&search,problem);
```

- Timeout for the whole problem

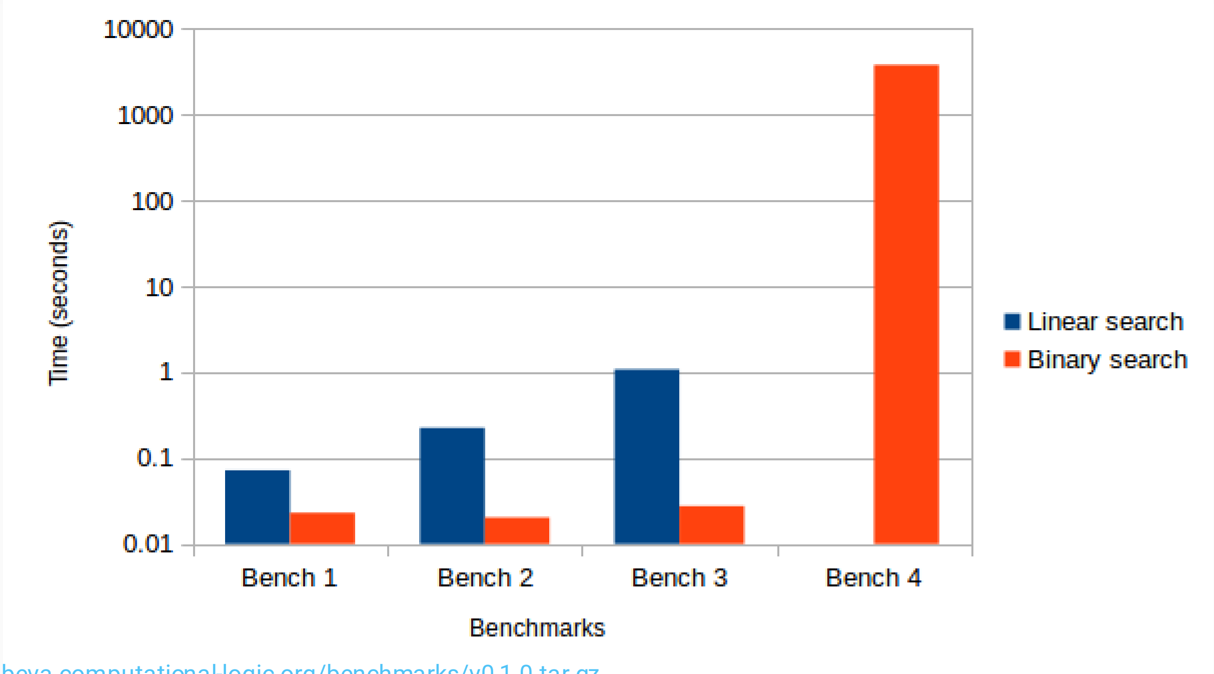




Architecture



Experiment



http://pbeva.computational-logic.org/benchmarks/v0.1.0.tar.gz

Conclusions & future work

Conclusions

- Goal of improving the overall time required to solve pseudo-Boolean minimisation problems
- A more user-friendly interface
- Two search algorithms to find the minimum value
- Two timeout strategies to finish the execution

Future work

- Github repository, documentation, installer and wiki
- Iteration 3: Multithreading
- Allow maximisation and other relational operators

Analysis of the planning

Timeline

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

Budget

Estimated budget

Direct costs	10.015,00
Indirect costs	140,73
Contingency	3.025,61
Unforeseen	816,14
Total	13.997,48

Real budget

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

Sustainability

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

Thank you for your time

Questions?

https://github.com/marcbenedi/SAT-TFG