Contents

[Abstract 2](#_Toc127039337)

[Background 2](#_Toc127039338)

[Boolean Satisfiability Problem (SAT) 2](#_Toc127039339)

[Conjunctive Normal Form 2](#_Toc127039340)

[Pre-Solvers 2](#_Toc127039341)

[SATfeatPy 2](#_Toc127039342)

[Objective 2](#_Toc127039343)

[Design 3](#_Toc127039344)

[Implementation 3](#_Toc127039345)

[Term Reducer 3](#_Toc127039346)

[Variable Selector 3](#_Toc127039347)

[Dataset Populator 3](#_Toc127039348)

[Random Forest 3](#_Toc127039349)

[Software eng issues 3](#_Toc127039350)

[Datasets 3](#_Toc127039351)

[Inductive Inference 3](#_Toc127039352)

[Better one? 3](#_Toc127039353)

[Variable Heuristics 4](#_Toc127039354)

[Conclusions 4](#_Toc127039355)

[Further Study 4](#_Toc127039356)

[References 4](#_Toc127039357)

# Introduction

The aim of this project was to construct a variable-based pre-solver – i.e. a pre-solver which generates partial assignments by iteratively computing variable level statistics and leverages them to decide which variable to next assign, and which assignment to give it.

# Literature Review

# Background

## Boolean Satisfiability Problem (SAT)

Boolean satisfiability problem (SAT problems) deal with instances of Boolean formulae. A set of atomic Boolean variables S={x1,…,xn}, are combined by the operators AND, OR, and NOT to form a composite statement which itself is a Boolean variable.

Typically, SAT solvers attempt one or more of the following:

* Discern whether an instance is satisfiable (SAT/UNSAT), i.e., does there exist an assignment of {0,1} to every variable in S for which the entire statement will evaluate to True?
* Find a solution, i.e., identify an assignment (or set of assignments) which satisfies the statement.
* Find a backdoor/backbone, i.e., a subset of variables in S which, given the correct assignment, will satisfy the instance regardless of the assignments of the other variables.

## Conjunctive Normal Form

Each variable x in a Boolean formula can appear in two forms, x and NOT x. These forms are known as literals, and for each literal (variable, assignment) pair, its opposite assignment is known as a negation.

A clause is a subset of the literals of S, combined with the operator OR to form a Boolean statement. (x3 OR x5 OR NOT x6) is an example of a valid clause. It follows that if at least one literal in a clause evaluates to True, the whole clause will be satisfied.

Conjunctive Normal Form (CNF) is the most common structure of SAT instances. It’s comprised of a number of clauses combined by AND statements. Every clause must be satisfied, but each clause requires only one valid literal assignment for satisfaction.

## Pre-Solvers

SAT solving is applied to scheduling, constraint solving, chip design, and many other fields. However, all of the above problems are NP-hard. There is great variability in the difficulty of solving SAT instances, even among statements of a similar size – difficulty depends not only on the size of the number of variables and clauses, but also how those clauses are composed.

Pre-solvers are a key technique used in SAT solving. They attempt to simplify SAT instances (by eliminating clauses or variables, re-writing clauses, completing partial assignments, and other means) before the instance is passed to the main SAT solving algorithm.

# Methods

## Design

I built something

## Implementation

### Term Reducer

* Large because we need info about variables
* Automated unit propagation

### Variable Selector

Majority of functionality is in function “run”.

### Dataset Populator

Here

### Random Forest

77% accurate with CBS dataset prior, 2/3rd skewed for SAT

2/3rd as trying to mimic IRL where the optimally performing variable selector will have either a true and a false branch, or both branches true.

Bad dataset due to skew towards SAT, poor decisions being made early on in process.

If anything there should be a skew for unsat.(but try balanced dataset)

Otherwise, penalise for specificity.

Plot ROC curves

### Software eng issues

Little mistakes:

* Type error (variable, is\_negation) comparing {1, -1} to {True, False}
* When populating dataset and writing to file, wrote after each file in case of crashing. Forgot to reset string to empty so was writing n(n+1)/2 lines instead of n lines for n files.
* Near beginning, forgot to check for presence in unit clauses when removing variables (silly)
* When randomly choosing variable, had off-by-one error. Variables are counted from 1 as 0, -0 doesn’t make much sense. Randomly choosing variables for dataset population used ‘random.randint(0, cnf.num\_variables-1)

Refactors:

Added removed attribute to clauses and variables. Did this instead of setting index to None. The (minimal) save of memory wasn’t worth conflating indices with removal status.

# Datasets

## Inductive Inference

Here

## CBS

Here

## Better one?

Here

# Variable Heuristics

The variable heuristic iterations are essentially increasingly complex generalisations of the pure literal rule [explain].

* Random
* Skew of positive to negative appearances
* Clause size introduced

# Conclusions

# Further Study

* Expand variable heuristic to include covariance matrix, BI graph, horn clause.
* Check purity of branch choices – do we go with the “purer” literal every time? If so the random forest is extraneous
* Expand to RL model with clause rewriting, subsumption checks, tautology checks, and possibly choice of solvers to use (like SATzilla).
* Add in backtracking – create two cutoffs, one for termination and one (higher) for checkpoint. At each step, save the current cnf if it is above the checkpoint threshold. Instead of terminating, return to checkpoint.

# References