SAT Solvers

Student Name: King Him Cheung

Supervisor Name: Dr. Friedetzky

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

Abstract —

Context/Background

The purpose of SAT Solvers is the solve the satisfiability problem, they are in use in modern day technologies for many practical reasons including: model checking, hardware verification, automatic test pattern generation, planning, scheduling, and even challenging problems from algebra. Over the years, we have seen new implementations of these SAT solvers which more importantly are becoming more efficient.

Aims

The aim of this project is to research modern day SAT solvers and understand the reasons to their efficiency, discovering different implementations and understanding their uses is therefore vital. Furthermore, another aim of this project and one in which concrete results can be produced is the implementation of a SAT Solver capable in solving more difficult satisfiability problems.

Method

Various SAT solvers should be researched, comparisons in their run times should be made and recorded. Implementation on different variations of SAT Solvers ranging from their complexity, understanding the underlying concepts of these implementations, taking into consideration the algorithm and data structures involved.

Proposed Solution.

Research variations of the SAT Competition solvers as a initial look into the various implementations, compare running time of these variations, taking into account search problem implementations and other solutions.
Using Python to create test implementations without consideration of advanced optimisations and then moving on to produce more optimised versions either using optimised frameworks or a more low level programming language such as C++.
Research modern technologies such as machine learning and how they may be used to produce more efficient solvers.

Keywords — SAT Solver, search problem, machine learning.

I INTRODUCTION

SAT Solvers come in many variations, this project is to understand the practical purpose of SAT Solvers, how SAT Solvers have developed over the years and how some variations are implemented. Concrete results can be gathered from comparing SAT solvers through their running times and use cases. Moreover, implementation of SAT Solvers of different variations including the use of more modern techniques will be made. Overall, this project will consider SAT Solvers from a theoretical view and implementations will be made from a theoretical approach on ideas from search problem and machine learning concepts.

Project Domain

The boolean satisfiability problem is defined as: "does there exist a assignment of variables that satisfies the given boolean formula?"

In our consideration of this problem we will focus on Boolean Formulas in Conjunctive Normal Form: A conjunction of clause where a clause is the disjunctions of literals.

- A literal is a variable of the negation of the variable: x or $\neg$x.
- A conjunction (X $\wedge$ Y) or ($X_1 \wedge X_2 \wedge ... \wedge X_n$) is True only when all variables are True.
- A disjunction (X $\vee$ Y) or ($X_1 \vee X_2 \vee ... \vee X_n$) is True only when at least one variable is True.

This problem is NP-complete and many other problems can be encoded as a Satisfiability problem therefore by having efficient solvers for this problem means being able to solve many other problems. Some real world applications for SAT solvers include: Model checking, scheduling and problem solving.

Model Checking is used for hardware verification and is a technique of checking some property of the hardware by describing hardware functionality as a model: a set of states S = $\{X_1, X_2, ..., X_n\}$ and transition relations between states $\tau(S_1, S_2), S_1 \in S$ and $S_2 \in S$. Some properties checked may include: Liveness, where a property eventually occurs e.g. used to check whether a response is eventually given or to check if cycles occur and Safety where a property always holds e.g. the reachability problem, to check whether or not a bad state can be reached.
Bounded model checking where a property is checked within some k steps can be solved by SAT solvers checking for satisfiability on the formula:

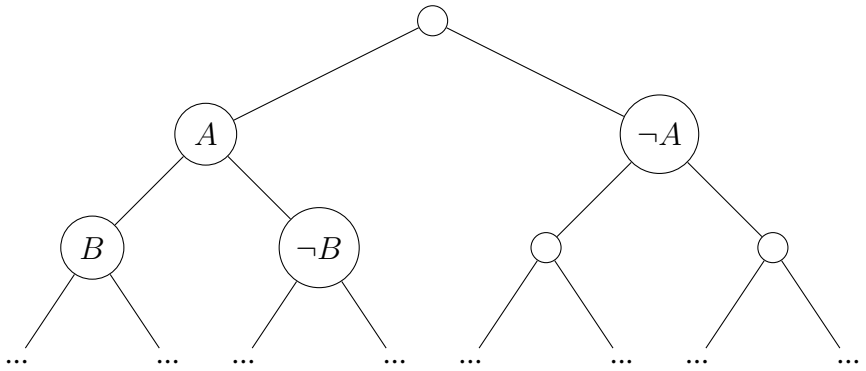
$$I(s_0) \wedge \bigwedge_{i=0}^{k-1} \tau(s_i, s_i + 1) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

Where $I(s_0)$ is defined as s_0 being the initial state, and all states s_i are iterated through by a legal transition state and none of the states reached have the property p.

Project Overview

The structure of this project will be focused on gaining insight in the multiple variations of SAT solvers from a theoretical view. Moreover, the projects goal of implementing of SAT solvers will allow a more in depth insight on workings of the algorithms.

DPLL algorithm - Davis Putnam Loveland Logemann - is one such variety and one that is a basis to many modern day solvers, the algorithms involves getting as input a CNF formula and outputs whether the formula is satisfiable and if it is in fact satisfiable then it returns an assignment of variables that does so. The procedure to this algorithm follows a tree where each node of the tree is a set of clauses S and at each node an assignment to a variable is made.



At each node after a variable has been assigned, we then continually apply inference (propagation) and so reducing the number of assignments and thus nodes in the tree. A node no longer has children if the node has the set $\{\}$ as its set of clauses or if the set of clauses contain the empty set $\epsilon \in$ S.
If a branch has $\{\}$ has its set of clauses then the formula is satisfiable and if all branches' set of all clauses contain the empty set $\epsilon \in$ S then the formula is unsatisfiable.

CDCL - Conflict Driven Clause Learning - is another variety of SAT solvers first used in Grasp [1] which has influenced many modern SAT Solvers, this variety of solvers is characterised by adding new clauses to its initial set of clauses as the algorithm progresses and non-chronological backtracking. As the search progresses and the branch of the search tree encounters a conflict i.e. a variable is previously set True and the most recent propagation requires the variable to be set False or vice versa. Through the conflict the algorithm learns the clause to be added to our set of clauses (clause database). The non-chronological backtracking and clause learning enables the algorithm to prune the search tree and significantly reduce the time of discovering a satisfiable assignment or to show that the formula is unsatisfiable.

There are also local search algorithms for SAT which is not complete and so may not provide an assignment of variables that satisfy the formula and may not be able to show that the formula is unsatisfiable however these algorithms can be shown to be efficient in specific cases of problems. Furthermore, a more modern approach to solving SAT problems is the use of machine learning and describing the problem as a classification problem, we can classify SAT problems so that solvers more efficient on specific categories of problems can be applied, this is in contrast to the more general SAT solvers.

Project Deliverables

Basic deliverables:

1. Implement a brute force SAT Solver using an intuitive programming language for the problem (may not be efficient)
    - - compromise on efficiency to produce a basic SAT solver using a high level language e.g. Python. Use an intuitive implementation relying on simple data structures.
2. Develop understanding on a programming language that aims to increase efficiency
    - - Read online documentation on a programming language that allows efficient use of data storage and manipulation e.g. C++, consider efficient uses of data structures and lower level data handling.
3. Discover and understand optimisations made to DPLL implementations of SAT Solvers
    - -Research and develop an understanding on faster algorithms and consider the implementation of such an algorithm in the high level programming language to show proof of concept.
4. Develop an understanding on machine learning
    - - Undergo an online course* on basic machine learning concepts

Intermediate Deliverables:

1. Research how machine learning can be applied to SAT Solving
    - - Read articles and research papers on the use of machine learning in SAT solving
2. Implement a DPLL SAT Solver using an efficient programming language
    - - implement DPLL Sat Solver using the previously studied lower level programming language.
3. Implement a DPLL SAT Solver involving multiple optimisations
    - - implement efficient data structures using the lower level language and implement researched optimisations to the algorithm e.g. watched literals.
4. Explore and implement other versions of SAT solvers e.g. local search

Advanced Deliverables

1. Analyse the efficiency benefits of the different optimisations in DPLL and its possible costs
2. Using the knowledge of most beneficial optimisations create a SAT Solver that is as efficient as possible (with the optimisations explored)
3. Implement a SAT Solver that utilises machine learning giving its benefits and costs

II DESIGN

This section presents the proposed solutions of the problems in detail. The design details should all be placed in this section. You may create a number of subsections, each focusing on one issue.

This section should be up to 8 pages in length. The rest of this section shows the formats of subsections as well as some general formatting information. You should also consult the Word template.

A Main Text

The font used for the main text should be Times New Roman (Times) and the font size should be 12. The first line of all paragraphs should be indented by 0.25in, except for the first paragraph of each section, subsection, subsubsection etc. (the paragraph immediately after the header) where no indentation is needed.

B Figures and Tables

In general, figures and tables should not appear before they are cited. Place figure captions below the figures; place table titles above the tables. If your figure has two parts, for example, include the labels "(a)" and "(b)" as part of the artwork. Please verify that figures and tables you mention in the text actually exist. make sure that all tables and figures are numbered as shown in Table 1 and Figure 1.

Table 1: UNITS FOR MAGNETIC PROPERTIES

Symbol	Quantity	Conversion from Gaussian

C References

The list of cited references should appear at the end of the report, ordered alphabetically by the surnames of the first authors. The default style for references cited in the main text is the Harvard (author, date) format. When citing a section in a book, please give the relevant page numbers, as in (**?**, p293). When citing, where there are either one or two authors, use the names, but if there are more than two, give the first one and use "et al." as in , except where this would be ambiguous, in which case use all author names.

You need to give all authors' names in each reference. Do not use "et al." unless there are more than five authors. Papers that have not been published should be cited as "unpublished" (**?**). Papers that have been submitted or accepted for publication should be cited as "submitted for publication" as in (**?**) . You can also cite using just the year when the author's name appears in the text, as in "but according to Futher (**?**), we . . . ". Where an authors has more than one publication in a year, add 'a', 'b' etc. after the year.