

# SATlab

## AE6102 - Parallel Scientific Computing and Visualization

**Student 1 Name:** Anmol Saraf

**Student 2 Name:** Nikhil Kaniyeri

**Roll No. 1:** 200070007

**Roll No. 2:** 200070050

**Instructor:** Prof. Prabhu Ramachandran

**Due Date:** 18<sup>th</sup> February, 2023

---

### Abstract

In computer science, the boolean satisfiability problem or the SAT problem is the procedure of automated theorem proving that can simplify a set of clauses. We propose to create a SAT Solver that uses classical methods like DPLL and CPCL combined with other tricks parallelly to create a faster unit solver. We also intend to apply this solver on use cases like graph network problems (circuit et al.) as well as visualization by implementing GUI for games like Sudoku and Minesweeper. Finally, we will attempt cryptanalysis on hashing by using inversion attacks on secure hash functions.

## 1 Introduction

The boolean satisfiability problem, or the SAT problem, is the problem of determining whether a solution exists that satisfies a given boolean formula. For example, the formula **A and B** has a valid solution, A being true and B being true, whereas **A and not A** has no valid solution. It is a significant problem in computer science and logic, as it is an NP-complete problem, in a group of various optimization problems. We propose to create a parallel solver for the above problem.

## 2 Methods of SAT solving

### 2.1 Unit Propagation

This method is also called the one-literal rule, since it is based on unit clauses. Because each clause needs to be satisfied, we know that this literal is true. Other clauses that contain this unit literal get simplified by the application of a few rules as described below:

- Any clause containing the literal is removed. (The clause is satisfied *as-is*.)
- In every clause that contains the negation of the literal, the literal is deleted. (The negation cannot contribute in it being satisfied.)

The two most common and efficient algorithms that build on this idea for solving SAT problems are known as the **Davis–Putnam–Logemann–Loveland** (DPLL) algorithm and the **Conflict-Driven Clause Learning** (CPCL) algorithm. These are defined as follows,

### 2.2 Davis–Putnam–Logemann–Loveland

DPLL is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form, i.e. if a boolean logic is a conjunction of one or more clauses. The DPLL algorithm for solving a SAT problem is as follows,

Algorithm DPLL

Input: A set of clauses  $\Phi$ .

Output: A truth value indicating whether  $\Phi$  is satisfiable.

```
function DPLL( $\Phi$ )
  while there is a unit clause  $l$  in  $\Phi$  do
     $\Phi \leftarrow \text{unit-propagate}(l, \Phi)$ ;
  while there is a literal  $l$  that occurs pure in  $\Phi$  do
     $\Phi \leftarrow \text{pure-literal-assign}(l, \Phi)$ ;
  if  $\Phi$  is empty then
    return true;
  if  $\Phi$  contains an empty clause then
    return false;
   $l \leftarrow \text{choose-literal}(\Phi)$ ;
  return DPLL( $\Phi \wedge l$ ) or DPLL( $\Phi \wedge \text{not}(l)$ );
```

This is a parallelly implementable algorithm but there is no learning from the wrong guesses made for the variables. The following algorithm CPCL takes account for that.

## 2.3 Conflict-Driven Clause Learning

The CPCL algorithm follows the same methodology as the DPLL algorithm but the back-jumping in CPCL is non-chronological. The main algorithm is as given below,

1. Select a variable and assign True or False. This is called decision state. Remember the assignment.
2. Apply Boolean constraint propagation (unit propagation).
3. Build the implication graph.
4. If there is any conflict:
  - (a) Find the cut in the implication graph that led to the conflict.
  - (b) Derive a new clause which is the negation of the assignments that led to the conflict.
  - (c) Non-chronologically backtrack ("back-jump") to the appropriate decision level, where the first-assigned variable involved in the conflict was assigned.
5. Otherwise continue from step 1 until all variable values are assigned.

This updates the memory but reduces the steps needed to solve the problem. This is a complex time and space trade-off algorithm.

## 3 Deliverables

We plan to create a solver to implement the algorithms parallelly and using this SAT solver, we will make a fast Sudoku with GUI. Furthermore, we will also attempt at making a Minesweeper game solver using the SAT solver. Finally, we will attempt cryptanalysis on hashed text by using inversion attacks on secure hash functions.

## 4 Timeline

Time Duration	Task to be Completed
26 Feb - 12 Mar	Learning and implementing algorithms parallelly
13 Mar - 26 Mar	Implementing Message Passing, to help concurrent threads from current executing threads
27 Mar - 10 Apr	Usage of solver on benchmarks; creating Fast Sudoku/Minesweeper with GUI
11 Apr - 26 Apr	Attempt Cryptanalysis on hash functions

## 5 Conclusion

We intend to apply both DPLL and CPCL algorithms and compare the serial (single thread) and parallel (multi-thread) performance and also measure the time/space trade-off. For application purposes we would simulate a Minesweeper like game built upon the satisfiability problem.

## 6 Links

### Github Repository

<https://www.github.com/kaniyeri/SATlab>

### References

[https://en.wikipedia.org/wiki/DPLL\\_algorithm](https://en.wikipedia.org/wiki/DPLL_algorithm)  
[https://en.wikipedia.org/wiki/Conflict-driven\\_clause\\_learning](https://en.wikipedia.org/wiki/Conflict-driven_clause_learning)  
[https://en.wikipedia.org/wiki/Implication\\_graph](https://en.wikipedia.org/wiki/Implication_graph)  
<https://eprint.iacr.org/2006/254.pdf>