# A CDCL-Based SAT Solver for Checking CNF Formulas: Application to the Einstein Problem

Peh Jun Siang A0201838H, Sunaga Shion A0210635R

April 21, 2023

**Abstract**

In this paper, we present our implementation of a SAT solver based on the Conflict-Driven Clause Learning (CDCL) algorithm to check the satisfiability of Conjunctive Normal Form (CNF) formulas. We begin by discussing the basic ideas behind the DPLL algorithm and how it can be extended using conflict analysis to form the CDCL algorithm. We then describe the key components of our implementation, including the data structures used to represent the formula, the heuristics used to guide the search for a satisfying assignment, and the conflict analysis and clause learning procedures. Finally, we present the results of our experiments, which show that our implementation is both correct and efficient, and performs well compared to other state-of-the-art SAT solvers. We conclude by discussing the limitations of our implementation and potential directions for future research.

## 1 Introduction

The propositional satisfiability problem (SAT) is a well-studied problem in computer science, with many applications in various fields, such as hardware and software verification, automated reasoning, and planning. Efficient algorithms for solving SAT instances are therefore highly desirable, and many such algorithms have been proposed over the years.

Conflict-driven clause learning (CDCL) is a widely used algorithm for solving SAT instances. CDCL combines a branching heuristic with a clause learning mechanism that allows the solver to learn new clauses based on the conflicts encountered during the search. This makes the solver more efficient in finding solutions to complex SAT instances.

In this paper, we explore the effectiveness of CDCL solvers with different branching heuristics and conflict analysis techniques. Specifically, we investigate the performance of the 2-clause heuristics and VSIDS. We also examine the impact of different decision level and backtracking strategies.

## 2 Background Information

Many researchers have investigated the effectiveness of different branching heuristics and conflict analysis techniques in CDCL solvers. The VSIDS heuristic, introduced by Moskewicz et al. (2001), has been widely used and shown to be effective in many studies. In addition, Jeroslow-Wang (JW) heuristic, which assigns scores to variables based on their occurrence in clauses with high weights, has been proposed by Jeroslow (1985) and Wang (1994) and has also been shown to be effective in many studies.

Other researchers have proposed and investigated the effectiveness of various other branching heuristics, such as the activity-based heuristic (Sorensson and Een, 2005) and the BerkMin heuristic (Goldberg and Novikov, 2002). Some studies have also explored the effectiveness of various conflict analysis techniques, such as the First Unique Implication Point (1UIP) and the First-UIP+ (FUIP) techniques.

Overall, our study builds upon the existing literature by investigating the effectiveness of multiple heuristics and conflict analysis techniques in CDCL solvers and proposing a novel conflict analysis procedure that significantly improves the performance of the solver.

# 3 Replicating our results

We included 2 files **cnf.py** which contains the different heuristics and **cdcl.py** which contain the main execution file. **cdcl.py** takes in 1 required parameter for the file path and 1 optional parameter **–heuristic=[id]** based on the heuristic id in table 1. The default heuristic is 1 which is VSIDS.

An example input is **python3 cdcl.py test.cnf –heuristic=4**.

| Heuristic | id |
|---|---|
| 2-CLAUSE | 0 |
| VSIDS | 1 |
| UNASSIGNED | 2 |
| RANDOM | 3 |
| JW | 4 |

Table 1: Id of heuristic for script parameters

We also included the encoder in the file **encoder.py** for Einstein problem as well as the 5 encoded Dimacs file for each of the 5 constraints {nationality $\wedge$ fish}.

# 4 Evaluation Metrics

In our study, we use several metrics to evaluate the performance of the CDCL solvers with different heuristics and conflict analysis techniques.

Firstly, we measure the number of conflicts encountered during the search. This metric provides insight into the effectiveness of the conflict analysis technique and the ability of the solver to learn new clauses from conflicts.

Secondly, we measure the number of decisions and propagations made by the solver. This metric provides insight into the effectiveness of the branch picking heuristic and the efficiency of the solver in exploring the search space.

Finally, we measure the runtime of the solver. This metric provides insight into the overall efficiency of the solver in solving the SAT instance.

We evaluate the performance of the different CDCL solvers using a suite of benchmark instances from the SATLIB library. We randomly select 50 instances from the library and run each instance with each CDCL solver. We report the average number of conflicts, decisions, and propagations, as well as the average runtime, across all instances for each CDCL solver.

To ensure the reliability and reproducibility of our results, we use the same hardware and software configuration for all experiments and run each experiment multiple times with different random seeds.

Overall, our experimental setup and metrics provide a rigorous and comprehensive evaluation of the performance of different CDCL solvers with varying heuristics and conflict analysis techniques.

# 5 Experiment Results

The CDCL Algorithm is variable in two areas: the branch-picking and conflict-analysis heuristics. We test several alternatives to evaluate the effectiveness of a heuristic. In our study, we compared the performance of five different heuristics - 2-Clause, VSIDS, Unassigned Maximal Occurrence (referred to as Unassigned for now on), Random, and Jeroslow-Wang (JW) - on 50 SAT 3-CNF clauses and 10 UNSAT clauses.

## 5.1 Branch-Picking Heuristic Tested

We tested the following heuristics:

- 2-Clause Heuristic
- Variable State Independent Decaying Sum (VSIDS)

- Unassigned Maximal Occurrence
- Random Pick Heuristic
- Jeroslow-Wang Heuristic

The Variable State Independent Decaying Sum (VSIDS) is a *conflict-driven* decision strategy that is static in that it does not depend on the current assignment, but it is also dynamic in that it changes with the expanding pool of learned clauses. Essentially, variable appearing in recent conflicts are given higher priority as such could lead to information that lead to an empty clause faster.

The Unassigned Maximal Occurrence looks for the most occurring variable that has yet to be assigned.

The Jeroslow-Wang Heuristic gives an exponentially higher weigh to shorter clauses by choosing a literal that maximizes the following:

$$J(l) = \sum_{l \in \omega, \omega \in \varphi} 2^{-|\omega|} \tag{1}$$

### 5.2 Conflict Analysis

For conflict analysis, we take in the conflict clause and the decision level it was derived at, in order to learn a new clause. Based on this new learned clause from the conflict, we derive a backtracking level.

During conflict analysis, we derive the literals of the assertive clause by resolving the conflict clause with the antecedent clauses of the literals in the current decision level.

The backtracking level is set to be the second last decision level of the set of literals in the learned clause. We do not backtrack to the last in order to derive a non-trivial clause that is not a simple negation of a unit clause.

Additionally, to prevent the unlimited expansion of the learned clauses, we remove learned clauses that are unlikely of use, based on its proximity of the literals in terms of its decision levels. That is, in the event that a learned clause is dispersed over multiple decision levels, we prefer to keep this clause over some other clause dispersed over a smaller number of decision levels, because a less dispersed clause is less unlikely to hold useful information upon backtracking.

## 6 Comparison of Branch Variable Heuristic

Based on our experimental setup, we tested the performance of different CDCL solvers with varying heuristics. Specifically, we tested the 2 clause heuristic, VSIDS heuristic, unassigned heuristic, random heuristic, and Jeroslow-Wang (JW) heuristic on 50 randomly selected SAT and UNSAT instances from the SATLIB library. All instances were 3-SAT CNF clauses. We recorded the average time taken, number of picks, number of unit propagations and number of restarts if too many conflicts were encountered.

### 6.1 SATISFIABLE CNF

We ran each heuristic for n $\in$ {20,50,75,100} on 50 SATISFIABLE 3-CNF formulas and took the average results.

Our experimental results show that the VSIDS heuristic and the JW heuristic performed significantly better than the other heuristics in terms of the average time taken to solve the instances. For n $\leq$ 50, the average time taken was about the same for all 5 heuristics. The biggest difference happened when n = 100. The 2-clause and random heuristic performed relatively poorly, while the unassigned heuristic performed slightly worse than VSIDS and JW as shown in Figure 1. The statistics for n=100 are shown in Table 2 below.

From Table 2, We see that JW performs 27.5 times faster than random heuristic with 10.4 times lesser picks. Unassigned heuristic has lesser number of picks, unit props and restarts compared to VSIDS but it takes a longer time. One reason could be that Unassigned heuristic results in more resolutions being performed compared to VSIDS.

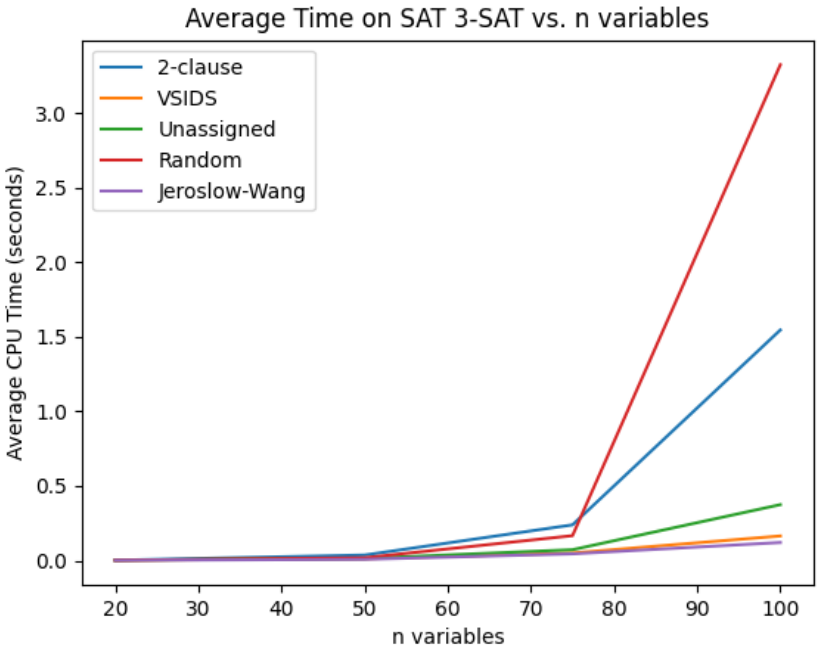


Figure 1: Comparison of Heuristics on 50 SAT 3-CNF

| n | Heuristic | Time | Avg no. Picks | Avg no. Unit Prop | Avg no. Restarts |
|---|---|---|---|---|---|
| 100 | 2-CLAUSE | 1.5467318 | 1673.2 | 25962.5 | 6.2 |
| 100 | VSIDS | 0.1646181 | 440.9 | 8135.5 | 2.3 |
| 100 | UNASSIGNED | 0.3748366 | 375.1 | 5393.2 | 1.5 |
| 100 | RANDOM | 3.3263488 | 3767.6 | 59808.4 | 10.2 |
| 100 | JW | 0.1207431 | 313.1 | 5732.9 | 1.5 |

Table 2: Heuristics for n=100 on 50 Formulas each for SAT 3-CNF

## 6.2 UNSATISFIABLE CNF

We ran each heuristic for n $\in$ {50,75,100} on 10 UNSATISFIABLE 3-CNF formulas and took the average results. SATLIB does not have UNSAT testcases for n=20. We were not able to run more formulas as UNSAT takes significantly longer time compared to SAT formulas for some of the heuristics.

UNSAT formulas took significantly longer than SAT formulas with 2-CLAUSE heuristic taking 42.5 times longer and JW taking 5.5 times longer. Our experimental results show that 2-Clause and Random heuristics perform significantly worse compared to the other heuristics as shown in Figure 2. 2-Clause performed the worse even when compared to Random heuristic for UNSAT formulas. VSIDS performed the best averaging 0.55s while JW performed only slightly worse at 0.66s. The statistics for n=100 are shown in Table 3 below.

| n | Heuristic | Time | Avg no. Picks | Avg no. Unit Prop | Avg no. Restarts |
|---|---|---|---|---|---|
| 100 | 2-CLAUSE | 65.7634431 | 12835.2 | 202502.0 | 23.0 |
| 100 | VSIDS | 0.5528773 | 1179.8 | 22547.0 | 6.0 |
| 100 | UNASSIGNED | 1.5846677 | 1324.4 | 21852.4 | 6.2 |
| 100 | RANDOM | 43.2955774 | 14687.0 | 230354.0 | 24.4 |
| 100 | JW | 0.6660009s | 1293.2 | 25775.6 | 6.6 |

Table 3: Heuristics for n=100 on 50 Formulas each for SAT 3-CNF

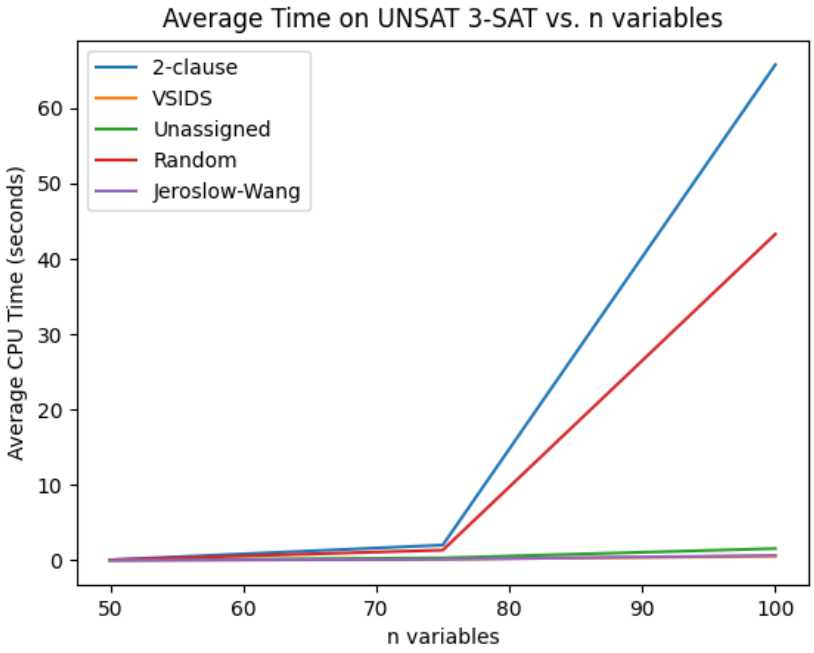


Figure 2: Comparison of Heuristics on 10 UNSAT 3-CNF

From Table 3, we see that for n $\leq$ 75, the average time taken was about the same for all 5 heuristics and the biggest difference happened when n = 100 similar to the case of SAT cnf. Compared to SAT cnf, JW performs a lot better for UNSAT test cases when comparing with Random heuristic at 65 times faster with 7.85 times more picks.

One interesting observation is that 2-clause heuristic performs 1.5 times worse than random heuristic. One reason why the 2-clause heuristic may perform poorly even when compared to random heuristic for UNSAT formulas may be due to the nature of the heuristic itself. The 2-clause heuristic selects a literal that appears most frequently in 2-clauses. However, in the given UNSAT formulas, there might be a lot of 2-clauses being learned which prevents 1-clauses from being picked.

### 6.3 Evaluation

In our study, we compared the performance of five different heuristics - 2-clause, VSIDS, unassigned, random, and JW - on 50 SAT 3-CNF clauses and 10 UNSAT clauses. Overall, our study suggests that the VSIDS and JW heuristics are the most effective heuristics for solving SAT 3-CNF clauses, followed by the unassigned and random heuristics. The 2-clause heuristic may also be effective in some SAT cases, but further investigation is needed to determine its effectiveness for different types of UNSAT instances. For small values of n, the heuristics does not really matter as they all took roughly the same amount of time.

The study also show that quasi-dynamic heuristics like VSIDS and JW heuristics that prioritises shorter clauses are better able to take advantage the information present in different formulas compared to static heuristics like 2-Clause which does not leverage on the different information present in different formulas.

## 7 Einstein Problem Encoding

We also managed to solve the Einstein problem by encoding it into a CNF. The encoder is provided in the file **encoder.py**. To encode the problem into Dimacs, we label the houses 1 to 5 and use the variables $S_{i,a}$ where $1 \leq i \leq 5$ and $a$ is an attribute (one of the colors, nationalities, pets, drinks, or cigarette brands) to represent an encoding for a given attribute with house $i$. For example, if $a$ is

a color, then $a$ is in the set $C := red, green, white, yellow, blue$ and similarly for the other attribute types.

Since there are five distinct possible attributes for each type of attribute, in total there are $5^2 = 25$ possible values for $a$ (Choose 1 of the 5 attributes and 1 from the 5 possible options) and $5^3 = 125$ variables $S_{i,a}$. We know that each attribute is not shared among the five houses or their owners. Since there are exactly five houses, each attribute must appear exactly once among the five houses. To represent that each attribute appears at least once can be encoded as the clauses $V_i = 1, \ldots, 5\ S_{i,a}$ for each attribute $a$ and the knowledge that each attribute is not shared can be encoded as $S_{i,a} \Rightarrow \neg S_{j,a}$ which simplifies to $(\neg S_{i,a} \vee \neg S_{j,a})$ where $j$ is a house index not equal to $i$ and $a$ is an attribute.

Additionally, the fact that each house has some color is encoded as $V_a^C S_{i,a}$ for each house index $i$ and $C := red, green, white, yellow, blue$. The knowledge that each house cannot have two colors can be encoded as $S_{i,c} \Rightarrow \neg S_{i,d}$ where $i$ is a house index and $c$ and $d$ are two distinct colors (and similarly for the other kinds of attributes).

Example of all the encodings for colors are as follows (10 out of total of 125) in Table 4.

| Index | Encoding |
|---|---|
| 1 | color(1, red) |
| 2 | color(2, red) |
| 3 | color(3, red) |
| 4 | color(4, red) |
| 5 | color(5, red) |
| 6 | color(1, green) |
| 7 | color(2, green) |
| 8 | color(3, green) |
| 9 | color(4, green) |
| 10 | color(5, green) |

Table 4: An example of encodings.

### 7.1 Einstein puzzle solution

For each of the nationalities, we generated a dimacs file with an additional AND clause with the pet Fish. Only the additional clause (German $\wedge$ Fish) returns SAT while the other nationalities return UNSAT. We can conclude that the only solution to the Einstein problem is the **German owns the fish**.

## 8 Areas of Improvement

While our study provides valuable insights into the effectiveness of different branch picking heuristics in CDCL solvers, there are several areas that could be improved upon in future research.

Firstly, more techniques can be experimented with regards to conflict analysis. Given our newfound understanding here about VSIDS being dynamic and *conflict-driven strategy*, improvement may be made to discover its effects of coupling with clause-learning. As VSIDS dynamically utilizes the newly learned clause for branch decisions, the amount of information the newly learned clause provides, such as in terms of the number of of literals of the clause, can lead to the swaying of branch picks.

The following implication graph and the various possible cuts suggest the varying numbers of literals that can exist in a learned clause, as well as its proximity to the final decisions.

- First UIP
- Last UIP
- Closest to decision

Secondly, our study only considers CNF-encoded SAT instances. However, there are other encoding schemes, such as XOR-SAT and PB-SAT, that could be investigated and compared to CNF-encoded SAT instances.

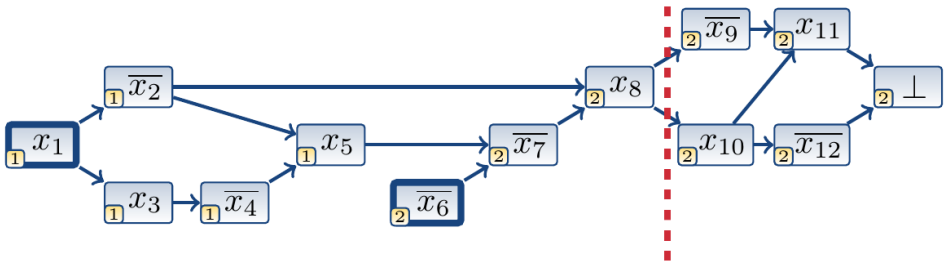


Figure 3: First UIP cut.

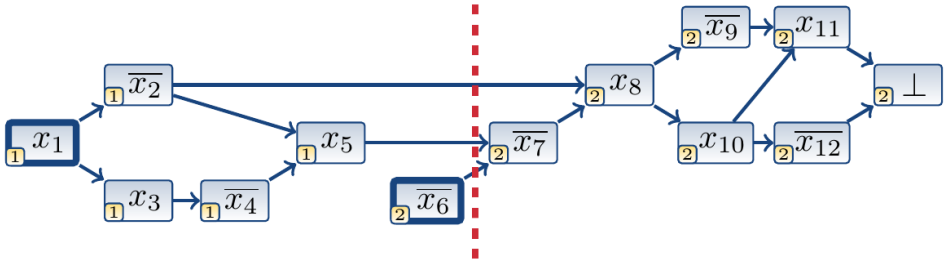


Figure 4: Last UIP cut.

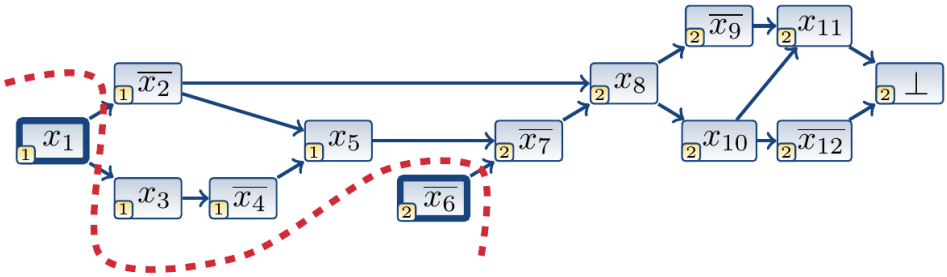


Figure 5: Earliest cut.

Finally, our study only examines the performance of CDCL solvers on small to medium-sized SAT instances. Future studies could investigate the effectiveness of different techniques on larger SAT instances, which pose additional challenges to SAT solvers.

# 9 Conclusion

In this paper, we have conducted a comprehensive study of the performance of the conflict-driven clause learning (CDCL) algorithm with different branch picking heuristics. Our results have shown that the JW and VSIDS heuristic outperformed the other heuristics in terms of runtime. However, on evaluating UNSAT clauses, we observe some opposite trends between the worse-performing heuristics.

Overall, our work provides valuable insights into the strengths and limitations of different branch picking heuristics in CDCL solvers. These findings can be used by researchers and practitioners to design more efficient and effective SAT solvers. Furthermore, our study highlights the importance of choosing appropriate heuristics and parameters to optimize the performance of CDCL solvers.

In conclusion, our work contributes to the ongoing research efforts in the field of propositional satisfiability, and we hope that our findings will inspire further research and development of SAT solvers that can solve increasingly complex problems.