Impact of Conflict-Driven Heuristics on SAT Solver Efficiency in Sudoku Puzzles

Student: 2843339, 2866238, and 2857122

Vrije Universiteit, Amsterdam, The Netherlands

Abstract. This report presents the development, design decisions, heuristics, experimental methods, and results for a SAT solver, aimed at addressing specific challenges in satisfiability problems. In this study, a large dataset of Sudoku puzzles was used to measure the performance of three SAT solvers: a basic DPLL algorithm as a baseline and two Conflict-Driven Clause Learning (CDCL) solvers using distinct heuristics, namely VSIDS and CHB. The two CDCL solvers use conflicts that arise from the solving process to add clauses that will limit the search space of a solution. This leads to a solution requiring a lower computational cost garnering an more efficient solve. While VSIDS prioritizes variable selection based on their frequency in recent conflicts, CHB uses a conflict history to guide its decisions. This study compares these heuristics to understand how their different strategies impact solver performance. Experimental results show that both CDCL heuristics significantly outperform the DPLL baseline, with CHB giving a 20% improvement in solving time compared to VSIDS, highlighting the value of quality over quantity of conflict management.

Keywords: SAT solver · Sudoku · Heuristics · Satisfiability. · DPLL · CDCL · VSIDS · CHB

1 Introduction

Sudoku is a puzzle game that has existed for around 2 centuries and became a pass-time enjoyed worldwide [4]. Despite its simple structure and small set of constraints, it has the ability to captivate players to play it for hours as they search for a solution. For those unfamiliar, Sudoku is typically played on a 9x9 grid with same cells pre-filled with numbers from 1-9 as initial clues. The objective is to fill in the remaining cells, ensuring that three main constraints are met: each row, each column, and each 3x3 subgrid (of which there are nine in the 9x9 grid) must contain each number from 1 to 9 exactly once.

Since Sudoku is a structural problem with defined constraints, it is an ideal candidate for a computer assisted solver such as a SAT solver. SAT solvers are designed to assess the satisfiability of a problem by determining whether the assignment of certain variables that are part of the problem can fit within the constraints imposed by the problem [5]. In the case of Sudoku, the variables represent the numbers placed in each grid cell, while the constraints are the rules

that must be followed to solve the puzzle.

Using a random or brute-force approach to obtain a valid configuration of variables is inefficient, as valuable information is gathered with each attempted configuration. When a variable is assigned a true or false value, that assignment can reveal information about which branches or assignments to avoid, thus reducing the search cost. This reduction is a primary objective of SAT solvers. The Davis-Putnam-Logemann-Loveland (DPLL) algorithm forms the foundation for this approach [6].The DPLL method solves SAT problems by recursive branching and backtracking using a systematic depth-first search. using techniques such as unit propagation to systematically reduce the search space and improve efficiency to find a solution [8].

An extension of the DPLL algorithm that we will explore further is the Conflict-Driven Clause Learning (CDCL) algorithm. In addition to using unit propagation and eliminating pure literals (literals that always appear with the same sign across all clauses), CDCL takes into account the conflict set. By analyzing the conflicts encountered during the search, CDCL can generate new clauses that capture this knowledge. The additional clauses can further limit the search space leading to a faster more efficient solve of the problem. CDCL also uses non chronological backtracking meaning it can jump back several decision levels to eliminate conflicting assignments [12].

Two of the most researched heuristics are the Conflict History-Based (CHB) heuristic and the Variable State Independent Decaying Sum (VSIDS). Both heuristics affect the solver's effectiveness and performance by controlling the choice of variables to branch on during search [10]. Liang and al., who proposed the CHB heuristic, demonstrated how competitive it is with VSIDS [9]. Moreover, the literature demonstrates that VSIDS and CHB have dominated the heuristics landscape in recent years, since almost all CDCL solvers that have been submitted in recent SAT races and competitions utilize variations of these approaches [3]. This observation promotes the following research question:

What effects do various CDCL heuristics have on SAT solver efficiency and performance?

For this research a comparison will be made between a basic DPLL algorithm and CDCL algorithm. The latter uses uses 2 variations of heuristics: CHB and VSIDS. By investigating the difference between the DPLL algorithm and CDCL algorithm as well as taking a closer look at what heuristic of the CDCL algorithm works best with Sudoku puzzles [11]. To evaluate how these heuristics affect solving power, the research experiment aims to answer the research question by measuring "efficiency" (such as time to solution, learned clauses, amount of implications) and "effectiveness" (such as success rate on a range of solved Sudoku puzzles (according to the rules of Sudoku).

[15]

1.1 Hypothesis

For this study we hypothesize that using the VSIDS heuristic in a SAT solver results in more decisions compared to the CHB. This outcome would indicate that VSIDS enables more decisions, leading to a more efficient solution process.

This notion is supported by the fact that recent conflict clauses, which are thought to be crucial in directing the search process through challenging SAT problems, are expected to be the primary focus of VSIDS. VSIDS uses recent conflict clauses to improves the performance of the solver leading to more decisions [11]. In contrast to CHB, which based choices on a larger historical record of variable performance, it is anticipated that VSIDS would need fewer decision points (i.e., variable assignments) and back jumps (i.e., retracing steps to a prior decision point). As a result, using VSIDS should result in more focused and informed decision-making, which will enhance performance, especially in terms of solver efficiency.

2 Design Decisions

The design decisions revolved around evaluating the performance of SAT solvers utilizing two distinct heuristics, namely CHB and VSIDS. DPLL was chosen as baseline for this study because of its simplicity. Due to the lack of optimizations on DPLL, it can be expected that heuristics will produce a higher performance. Our implementation of this particular algorithm makes use of unit propagation and backtracking to implement depth-first search. No heuristics were incorporated for this algorithm.

CDCL (Conflict-driven clause learning), on the other hand, was chosen due to its conflict driven approach, since both CHB and VSIDS are conflict driven heuristics. CDCL is based on DPLL, which makes it suitable for the experiments performed during this study.

Non chronological backtracking and optimization techniques like watched literals are common features in state-of-the-art CNF-based sat solvers [1]. For this reason, the CDCL algorithm implemented for this study employs both techniques. Back jumps are used to backtrack to the decision level that caused a conflict.

The watched literal technique consists on watching only two literals from every clause, updating these watches when one or both is set to false. If a pair of non-falsified watches cannot be found, a literal will be propagated, in case the clause has only one literal. Otherwise a conflict will be indicated [7]. This technique relies on the fact that, in SAT solvers, a clause is considered satisfied

as long as at least one of its literals evaluate to true. When using watched literals, it is not necessary to repeatedly evaluate every single literal in a clause to determine its satisfiability. This leads to performance gains, especially when considering long clauses.

3 Heuristics

Heuristics direct the variable selection procedure in SAT solvers during the search, which has a substantial effect on how effective and efficient the solver is. While they both aim to give variables top priority when making decisions, their processes and underlying presumptions on the basis of conflicts are different [2].

3.1 Conflict History-Based (CHB)

The Conflict History-Based branching heuristic (CHB) uses the history of conflicts that have been encountered throughout the solution process to dynamically choose variables for branching in a SAT solver. CHB sets a Q score to 0 initially. The Q score, that is expressed in equation 1, is described as a floating point that CHB keeps track of for every boolean variable.

$$Q[v] = (1 - \alpha)Q[v] + \alpha r_v \tag{1}$$

This score is modified depending on the variable's contribution to conflicts during the problem-solving process. The Q score is then updated every time the variable is involved in a conflict, and this is based on how recent it appeared in conflict analysis. The Q score update is influenced, however, by a reward value(r_v). Equations 2 shows the reward value is calculated respectively.

$$r_v = \frac{multiplier}{numConflict - lastConflict + 1} \tag{2}$$

Depending on whether the variable causes a conflict after being propagated, the multiplier in this case can be either 1.0 or 0.9. This configuration promotes the usage of variables that have more recently led to conflicts. The constants used in this study were derived from [9] where α updates the Q scores by a step-size starts at 0.4 by default, then decreases over time to a minimum of 0.06 after each conflict, allowing α CHB to adapt as the the search progresses, [14].

3.2 Variable State Independent Decaying Sum (VSIDS)

VSIDS was first introduced as part of the Chaff SAT solver [11] and became popular due to its efficiency. Because of that, many variations of this heuristics where proposed over time. For this experiment, a version of VSIDS based on MiniSat [13] was implemented.

This heuristic operates by assigning an activity score for each variable in every polarity, based on their involvement in conflicts. The activity scores for each

variable present in the conflicting clause has its score increased. Activity scores are decayed by a given decay factor every time a conflict is detected. This ensures that priority will be given to variables involved in conflicts more often. A decay factor of 0.95, the same factor used by MiniSat, is used for this heuristics during this study.

During variable selection, VSIDS chooses the unassigned variable with the highest activity score. This ensures that variables that appears more often in conflicting clauses are picked earlier in the solve process. This approach helps to guide the search towards areas that are more likely to cause conflicts. This could lead to a reduction in the amount of backtracks and decisions.

4 Experimental Design

Two different algorithms where used to evaluate the effects of using CHB and VSIDS heuristics. As described in Section 2 a basic version of DPLL, without any heuristic, is used to set a baseline. The CDCL algorithm is used to test both heuristics selected for this study. This algorithm was implemented in a modular way, allowing the heuristics to vary independently without requiring any changes to general CDCL algorithm. This approach allowed us to test the effects of the heuristics alone, without the risks of introducing human errors during implementation phase. This was achieved through the use of dependency injection for this particular algorithm.

Each of the three versions of SAT solvers was tested in a multi-threaded, automated environment using a dataset of 3,614 unique 9x9 Sudoku puzzles. To ensure the correctness of the solutions, all outputs generated by the SAT solvers were validated to confirm they represented valid Sudoku solutions.

During experimentation phase, the following metrics were collected for both DPLL and CDCL:

- **Number of decisions:** Indicates the amount of explicit choices the solver made. Explicit choice in this case means assigning *true* or *false* to a given variable.
- **Number of implications:** Amount of times the solver deducted the value (*true* or *false*) of a variable.
- **Number of conflicts**: Counts how many times clauses presented contradictions.
- **Elapsed CPU time:** Amount of time the SAT solver took to solve a Sudoku puzzle.

In addition to the metrics mentioned above, the frequency in which clauses were learned was also collected for CDCL specifically.

5 Experimental Results

Based on the experimental setup described in Section 4 the performance of the 3 different SAT solvers is analyzed below. Using the metrics of the conflicts, decisions, implications and elapsed CPU time it is possible to evaluate whether the results are what we expected and what new information was found relating to our research question.

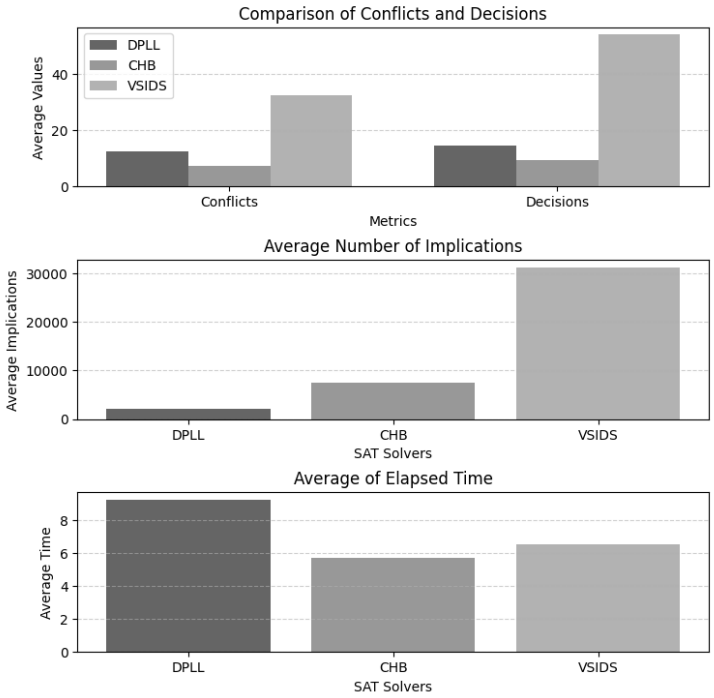

Fig. 1. General overview

When comparing the three different SAT solvers, it is evident which SAT solver family worked most efficiently. Observing figure 3, the average time taken to solve a Sudoku for the DPLL algorithm is twice as much as the CDCL sat solvers. This is due to CDCL learning new clauses from conflicts, which effectively reduces the search space, thus shortening the time needed to find a solution. This learning mechanism provides CDCL solvers with a significant advantage over the rudimentary DPLL approach.

Another metric to observe is the number of conflicts for each solve per SAT solver. Figure 1 shows that the number of conflicts for VSIDS was the highest,

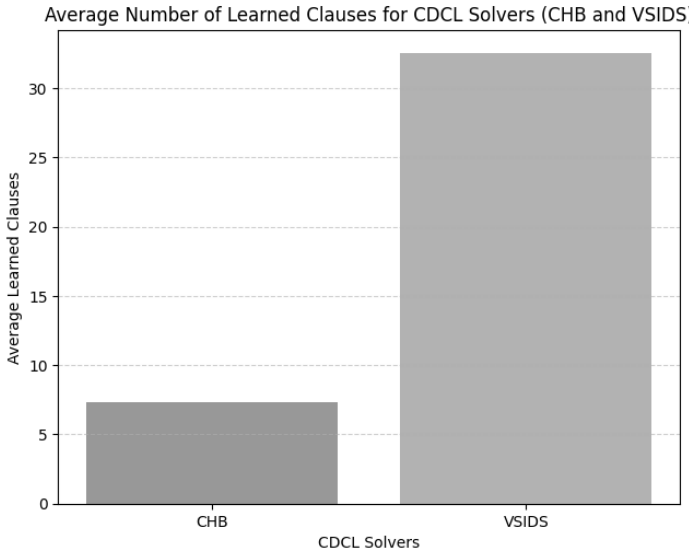

Fig. 2. Average number of learned clauses (VSIDS vs. CHB)

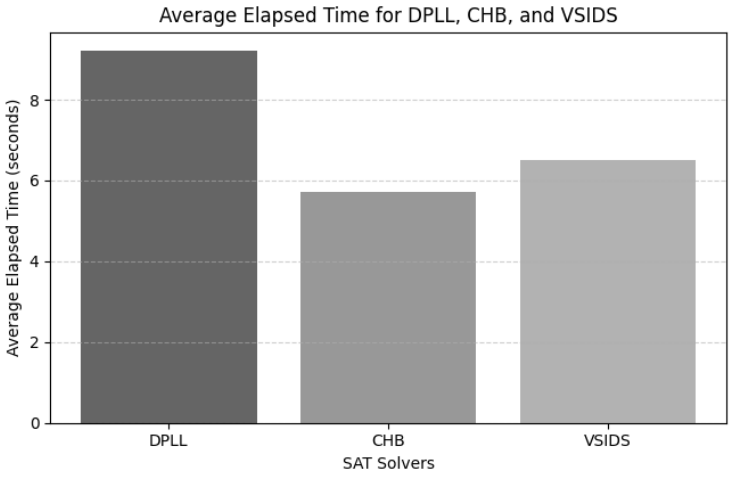

Fig. 3. Average elapsed time for each solver

followed by DPLL, and then CHB. This could be due to VSIDS's greedy search strategy, as it focuses on variables that have had a conflict recently. This means it will attempt to exhaust certain branches if the variables involved in the branch are contributing to more conflicts, thus increasing the number of learned clauses. Alternatively, CHB does not share a similar strategy, as it uses the entire conflict

history of each variable to decide which one to pick next.

Delving deeper into the differences between CDCL solvers, Figure 2 shows that VSIDS generated significantly more learned clauses than CHB during each solve (7 compared to 38). However, the elapsed time to solve Sudoku's between the two was minor (only 0.2 to 0.3 seconds over 3,000+ Sudoku puzzles, as shown in Figure 3). This indicates that the CHB's strategy of prioritizing clause quality over quantity is indeed more efficient than the VSIDS strategy, albeit just by 20 percent in terms of the elapsed CPU time. The key insight is the use of different heuristics leads to the use of different learning strategies. VSIDS emphasizes quantity in learned clauses, while CHB balances fewer clauses with greater impact.

The behavior of conflicts in all three sat solvers can also be reflected in the number of implications measured. The number of implications in VSIDS compared to CHB and DPLL can be explained by VSIDS's aggressive conflict-driven strategy. Each conflict in VSIDS leads to a re-evaluation of variable assignments, often triggering a cascade of further implications. In essence, every conflict generates additional inferences about variable values, reducing the search space. In contrast, CHB does not focus as intensely on recent conflicts and has a more balanced approach, leading to fewer implications. DPLL, being a simpler algorithm, generates fewer implications overall.

6 Conclusion

To conclude it is clear that the CDCL based algorithms had a better efficacy of solving Sudoku's than DPLL algorithm. This was due to the quality of learned clauses of CHB and quantity of learned clauses of VSIDS heuristics. Since the DPLL does not learn from conflicts, its performance is used on areas of the search space that were otherwise overlooked by the CHB and VSIDS.

By using clauses learned by conflicts, both CHB and VSIDS were able to solve Sudoku puzzles on average twice as quick as the baseline DPLL algorithm. A notable difference between the strategies of these two CDCL heuristics lies in their approaches to clause learning. VSIDS had an aggressive clause learning strategy, collecting a large substantial number of clauses in a short amount of time compared to CHB and DPLL. This strategy was performed better than the DPLL algorithm however it was slightly outperformed by the CHB heuristic. CHB's strategy solving process had a 20 percent performance boost in terms of average time to solve Sudoku puzzles compared to VSIDS.

The hypothesis states that using the VSIDS heuristic in a SAT solver would result in a more efficient solve compared to CHB. This was due to the theory that VSIDS enables more decisions. Although this is correct as VSIDS did enable more decisions than CHB, due to the similar performance as CHB, it can

be concluded that CHB made fewer but more informed decisions. Hence we can reject hypothesis and conclude CHB led to a more efficient solving process.

References

1. Biere, A., Heule, M., van Maaren, H.: Handbook of satisfiability, vol. 185. IOS press (2009)
2. Cai, S., Zhang, X., Fleury, M., Biere, A.: Better decision heuristics in cdcl through local search and target phases. Journal of Artificial Intelligence Research **74**, 1515–1563 (Aug 2022). https://doi.org/10.1613/jair.1.13666, http://dx.doi.org/10.1613/jair.1.13666
3. Cherif, M.S., Habet, D., Terrioux, C.: Combining VSIDS and CHB Using Restarts in SAT. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming (CP 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 210, pp. 20:1–20:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.CP.2021.20, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2021.20
4. Chi, E.C., Lange, K.: Techniques for solving sudoku puzzles. arXiv preprint arXiv:1203.2295 (2012)
5. Claessen, K., Een, N., Sheeran, M., Sorensson, N.: Sat-solving in practice. In: 2008 9th International Workshop on Discrete Event Systems. pp. 61–67 (2008). https://doi.org/10.1109/WODES.2008.4605923
6. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (Jul 1960). https://doi.org/10.1145/321033.321034, https://doi.org/10.1145/321033.321034
7. Devriendt, J.: Watched propagation of-integer linear constraints. In: International Conference on Principles and Practice of Constraint Programming. pp. 160–176. Springer (2020)
8. Fredrikson, M., Martins, R.: Lecture notes on sat solvers & dpll (2023)
9. Liang, J., Ganesh, V., Poupart, P., Czarnecki, K.: Exponential recency weighted average branching heuristic for sat solvers. Proceedings of the AAAI Conference on Artificial Intelligence **30**(1) (Mar 2016). https://doi.org/10.1609/aaai.v30i1.10439, https://ojs.aaai.org/index.php/AAAI/article/view/10439
10. Marques-Silva, J., Sakallah, K.: Grasp: a search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**(5), 506–521 (1999). https://doi.org/10.1109/12.769433
11. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: Proceedings of the 38th annual Design Automation Conference. pp. 530–535 (2001)
12. Nadel, A., Ryvchin, V.: Chronological backtracking. In: Theory and Applications of Satisfiability Testing–SAT 2018: 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9–12, 2018, Proceedings 21. pp. 111–121. Springer (2018)
13. Sorensson, N., Een, N.: Minisat v1. 13-a sat solver with conflict-clause minimization. SAT **2005**(53), 1–2 (2005)
14. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction, ser. adaptive computation and machine learning (1998)
15. Weber, T.: A sat-based sudoku solver. In: LPAR. pp. 11–15 (2005)