The question of the perfect SAT solving algorithm

Tănase Vlad Mihai

West University of Timișoara

May 11, 2025

Supervisor: Prof. Crăciun Adrian

Faculty of Mathematics and Computer Science,

Artificial Intelligence

Contents

The question of the perfect SAT solving algorithm

Tănase Vlad Mihai[†]

May 11, 2025

Abstract

SAT, or Boolean Satisfiability was the first ever NP-complete problem. It demands efficient algorithms for practical applications in automated systems responsible for reasoning. For the three main SAT solving algorithms, those being Resolution, Davis-Putnam and Davis-Putnam-Logemann-Loveland (which will be shortened to DP and DPLL respectively for coherence), this thesis will analyze their mechanisms of varying complexities by comparing each one.

[†] Student at West University of Timișoara, majoring in Artificial Intelligence

I. INTRODUCTION

The Boolean Satisfiability Problem (SAT) is a fundamental problem in computer science, asking whether there exists an assignment of truth values to variables in a propositional logic formula that makes the formula true. Specifically, SAT determines if a formula in Conjunctive Normal Form (CNF)—a conjunction of clauses, each a disjunction of literals—has a satisfying truth assignment. Proven NP-complete by Cook in 1971 (1), SAT holds significant theoretical and practical importance, serving as a cornerstone in complexity theory with applications in automated reasoning, model checking, planning, circuit design, and software verification. Given SAT's NP-completeness, the choice of algorithmic steps—particularly in search-based methods—greatly impacts performance. For instance, variable selection strategies in DPLL can drastically alter efficiency. This paper/thesis presents a theoretical and experimental comparison of three key SAT-solving algorithms: Resolution, Davis-Putnam (DP), and Davis-Putnam-Logemann-Loveland (DPLL). We examine DPLL's implementations with various heuristics, test them on diverse SAT instances, and analyze their performance in terms of computation time and decision steps, emphasizing how algorithmic choices influence outcomes.

II. BACKGROUND

SAT involves determining the satisfiability of a formula in Conjunctive Normal Form, which is a conjunction of clauses where each clause is a disjunction of literals. A formula is satisfiable only if there exists an assignment which turns all clauses true.

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, 1971

III. THEORETICAL COMPARISON

III.1. Resolution

Resolution is a refutation-based proof system for propositional logic, introduced by Robinson for automated theorem proving (2). For SAT, it tests unsatisfiability by applying the resolution rules: $(C_1 = (x \vee A))$ and $(C_2 = (\neg x \vee B))$, derive $(C = (A \vee B))$. The process continues until the empty clause is derived (indicating unsatisfiability) or no new clauses can be generated. Resolution is sound and complete but impractical for large instances due to its worst-case exponential time and space complexity, as some formulas require exponentially long proofs method based on rules from automated theorem proving (3).

- **Complexity:** Worst-case exponential time and space. Some formulas require exponentially long refutations.
- **Strengths and weaknesses:** While the method is sound and complete for unsatisfiability proofs, it is impractical for large cases because of *clause proliferation*.

III.2. Davis-Putnam method

THe method was introduced by Martin Davis and Hilary Putnam in 1960 (4). DP eliminates variables via resolution. For a chosen variable it resolves all pairs of clauses differing on that variable, adds the resolvents and then removes the original clauses, repeating until the formula simplifies to a trivial case. Even though the method is a complete solution, DP's space complexity rises as new clauses accumulate, making it unsuitable for large cases.

- **Complexity:** Exponential space and time due to clause generation.
- **Strengths and weaknesses:** Complete solution but impractical for large cases due to memory demands.

III.3. Davis-Putnam-Logemann-Loveland method

The method was developed in 1962 by Martin Davis, Hilary Putnam, George Logemann and Donald W. Loveland is a complete, backtracking-based search algorithm for deciding

the satisfiability of a propositional logic formulae in conjunctive normal form (5). It is a refinement of the earlier Davis-Putnam method. The key optimizations over DP are *unit propagation* (where if a clause has one literal, assignment is made to make the clause true, propagating effects) and *pure literal elimination* (where variables are assigned appearing only positively or negatively to satisfy all their clauses). The algorithm of the method follows the following steps:

1. Selects and unassigned variable.
2. Assigns it a boolean value (either *true* or *false*).
3. Simplifies the formula.
4. If empty it returns satisfiable, while if an empty clause appears, it backtracks.
5. Repeats until the formulae is solved.

- **Complexity:** Exponential time in the worst case but efficient with heuristics.
- **Strengths and weaknesses:** Its performance depends heavily on the variable selection.

[2] J. A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, vol. 12, no. 1, pp. 23–41, 1965

[3] A. Haken, "The intractability of resolution," *Theoretical Computer Science*, vol. 39, no. 2–3, pp. 297–308, 1985

[4] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, 1960

[5] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962

IV. EXPERIMENTAL SETUP

IV.1. Implementation

Resolution, DP and DPLL have been implemented in the C++ programming language due to the language's high degree of flexibility, space efficiency and fast execution time. The program will be compiled and run on the system terminal using MinGW's g++ command, with the general compilation flag for initialization of file specific solver executable " `g++ -O3 -march=native file.cpp -o solver.exe` ", instead of a common IDE in order to maximize true performance and get the most accurate results. Thus optimization is maximized, debug-only behaviour is disabled and all CPU-specific features are utilized allowing C++'s property of to-machine-code-compilation to be fully employed. For efficiency in the case of DPLL, unit propagation and pure literal elimination have been implemented. Three variable selection strategies were tested to explore the impact of choice in NP-complete problems:

1. **Dynamic frequency:** At each decision point, the program selects the unassigned variable appearing most frequently in the current set of unsatisfied clauses, similar to the Dynamic Largest Individual Sum (DLIS) heuristic (6).

2. **Random:** Selects a random unassigned variable, serving as baseline.

3. **Static frequency:** Orders variables by their frequency in the original formula (number of clause appearances) and selects the next unassigned variable in this order.

IV.2. Test instances

The algorithms have been evaluated on 3 types of SAT instances to capture diverse problem characteristics:

- **Random 3-SAT:** One instance with 20 variables and clause-to-variable ratios of 3.5 (underconstrained, 70 clauses), 4.15 (near phase transition, 83 clauses, as provided (7)), and 5.0 (overconstrained, 100 clauses), with 10 instances per ratio.

- **Structured instances:**

- *Pigeonhole Principle (PHP-5):* Encodes the problem of placing 6 pigeons in 5 holes, known to be unsatisfiable and challenging for methods based on resolution (3).
- *3-coloring:* A small graph (such as a graph of 5 nodes) encoded as a satisfiable SAT instance to test heuristic performance on combinational problems.

IV.3. Metrics

Three performance metrics were measured to asses heuristic impact:

- **Computational time:** Execution time, measured as wall-clock time, has been calculated through implementation of the C++ *<chrono> library* at program level.
- **Memory usage:** Memory usage has been calculated using an OS specific active process monitoring software: *Task Manager - Microsoft Windows*; and *Valgrid's Massif heap memory usage profiling tool - Linux*

[3] A. Haken, "The intractability of resolution," *Theoretical Computer Science*, vol. 39, no. 2–3, pp. 297–308, 1985

[6] J. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999

[7] D. Mitchell, B. Selman, and H. Levesque, "Hard and easy distributions of sat problems," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 459–465, 1992

V. RESULTS AND ANALYSIS

V.1. Random 3-SAT results

The following table summarizes the average performance across 5 instances per clause-to-variable ratio for DPLL with each heuristic, DP and Resolution. Times are in seconds and decision / propagation steps are counts.

V.2. Structured Instances

For the 3-coloring problem with medium size and complexity instances (satisfiable and unsatisfiable) as inputs, executions among all frequencies had a runtime of 0 seconds, implying a very negligible time difference, though with regards to their step count there are discrepancies suggesting varying efficiency: like in finding unsatisfiability - dynamic frequency (decisional steps: 22, propagation steps: 79) has a higher total step count than random selection (decisional steps: 20, propagation steps: 53) and static frequency(decisional steps: 28, propagation: 52). Thus the random heuristic appears to be more efficient in this scope despite the dynamic's academically accepted superiority. Still, the differences are negligible and the nature of the heuristic has virtually no impact on the algorithm's actual performance.

For PHP-5 (the Pigeonhole Principle problem) also with medium size and complexity instances, there were differences between heuristics in terms of execution time as well, them maintaining the near instant speed of 0 seconds. The efficiency of each frequency again boils down to their step count across unsatisfiability and satisfiability scenarios. Here, the static frequency is clearly inferior to its counterparts, managing step counts almost 3 times as high as the others (for unsatisfiability, decisional steps: 1444, propagation steps: 4122; for satisfiability, decisional steps: 44, propagation steps: 49). Therefore the static heuristic is clearly inferior in this scope as it's commonly considered in the academic space. Still, the differences are negligible and the nature of the heuristic again has virtually no impact on the algorithm's actual performance, though the static frequency should be avoided in large scale applications as a safe precaution. Memory usage was under 1 mb across all runs.

	Dynamic		Random		Static	
	3-coloring	PHP-5	3-coloring	PHP-5	3-coloring	PHP-5
unsatisfiability	Decision: 22 Propagation: 79	Decision: 660 Propagation: 1630	Decision: 20 Propagation: 53	Decision: 424 Propagation: 1482	Decision: 28 Propagation: 52	Decision: 1444 Propagation: 4122
satisfiability	Decision: 2 Propagation: 10	Decision: 11 Propagation: 18	Decision: 8 Propagation: 9	Decision: 15 Propagation: 20	Decision: 12 Propagation: 8	Decision: 44 Propagation: 49

The two structured instances have additionally been tested across the three main SAT solving algorithms in order to compare solving time. While DPLL is virtually instantaneous, DP struggles to reach the same speeds, lacking significantly at solving unsatisfiability for PHP-5 (0.85 seconds). While DP lags behind DPLL, Resolution is clearly the most inefficient and poorly optimized, reaching excessively long runtimes for even instances of medium complexity. The method also has an issue with memory usage due to its exponential space requirements (3).

	DPLL		DP		Resolution	
	3-coloring	PHP-5	3-coloring	PHP-5	3-coloring	PHP-5
unsatisfiability	0 s	0 s	0.0012 s	0.85 s	excessive runtime	excessive runtime
satisfiability	0 s	0 s	0.0013 s	0.001 s	excessive runtime	excessive runtime
	low memory usage, under 1 mb				excessive memory usage	

For a random 3-SAT instance input of large proportions, DPLL managed an execution time of 0 seconds, while DP managed one of 1.9 seconds. As with the Resolution method, the program exhibited a prolonged runtime which indicates suboptimal algorithm design, with RAM usage skyrocketing up to **250 mb**.

DPLL	DP	Resolution
0 s	1.9 s	excessive runtime
found satisfiability		couldn't find satisfiability cause: runtime error
low memory usage under 2 mb		excessive memory usage 200+ mb

V.3. Discussion

The dynamic frequency heuristic's superiority aligns with SAT's NP-completeness, where adaptive choices reduce the search space. Static frequency, while better than random, is proof of how important the need for informed heuristics actually is. Static frequency's poor

performance fails to account for formula changes during execution, its lack of dynamic solving abilities being key to its inferiority. Resolution and DP's theoretical space complexity ($O(2^n)$) contrasts with DPLL's linear memory usage, explaining their exclusion from practical implementation. These findings are consistent with studies on heuristic-driven SAT solving (6).

[3] A. Haken, "The intractability of resolution," *Theoretical Computer Science*, vol. 39, no. 2–3, pp. 297–308, 1985

[6] J. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999

VI. CONCLUSION

This work shows that the DPLL algorithm, when combined with dynamic variable selection heuristics like DLIS, is superior to Resolution and Davis-Putnam in real-world SAT solving. Though Resolution and DP are theoretically efficient, their exponential space requirement makes them unsuitable for big problems, as exemplified by their bad performance on structured problems like the Pigeonhole Principle. On the other hand, the effectiveness of DPLL, particularly as guided by adaptive heuristics, suggests the role of algorithmic choice in computation of NP-complete problems.

Even so, while better than some previous methods, DPLL is not to be regarded as perfect. Its worst-case time complexity is still exponential, a fundamental limitation imposed by the NP-completeness of SAT. In addition, its performance relies considerably on heuristics selection, as suggested by the less-than-optimal performances of static frequency tactics compared to dynamic or randomized ones. Computational constraints further restrict the potential of DPLL. Even with optimization, conventional computation architectures can never be efficient enough for truly scalable SAT solving. So, while DPLL is currently the pinnacle of conventional SAT solvers, the quest for a more powerful, all-environment algorithm goes on.

VII. Future Work

Future work could explore modern extensions like Conflict-Driven Clause Learning (CDCL), as discussed in (8), or parallel SAT solving to further enhance performance. Yet, the limitations of classical SAT solvers like DPLL invite exploration into alternative paradigms, particularly the intersection of traditional and quantum computing. A promising direction would involve a rigorous comparison between DPLL and quantum-based SAT algorithms, such as those leveraging Grover's search or quantum annealing. While quantum approaches offer theoretical speedups, their practical implementation is still growing, and a systematic evaluation could clarify their viability for real-world NP-complete problems.

Further research could also investigate hybrid systems, where classical DPLL is enhanced with quantum subroutines to mitigate the limitations of current quantum hardware. Beyond quantum computing, extending this work to include modern DPLL variants like Conflict-Driven Clause Learning (CDCL) solvers would provide valuable insights into the evolution of SAT-solving techniques. Finally, the development of adaptive heuristics—capable of dynamically adjusting to problem structure—could further bridge the gap between DPLL's current capabilities and the ideal of a truly universal SAT solver.

[8] A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds., *Handbook of Satisfiability*, vol. 185. IOS Press, 2009

References

S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, 1971.

J. A. Robinson, "A machine-oriented logic based on the resolution principle," *Journal of the ACM*, vol. 12, no. 1, pp. 23–41, 1965.

A. Haken, "The intractability of resolution," *Theoretical Computer Science*, vol. 39, no. 2–3, pp. 297–308, 1985.

M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, 1960.

M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

J. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

D. Mitchell, B. Selman, and H. Levesque, "Hard and easy distributions of sat problems," in *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 459–465, 1992.

A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds., *Handbook of Satisfiability*, vol. 185. IOS Press, 2009.

All source code for the SAT solving algorithms discussed in this paper is publicly available at: `https://github.com/tanasevladmihai/SAT_solving_main_algorithms`.