Analysis of Phase Transitions in 3SAT

Team:ae ae dee
AAD Project Bonus

December 1, 2025

1 Introduction

The 3-SAT problem is one of the most famous challenges in computer science. It asks whether we can find a way to assign "True" or "False" to a set of variables so that a specific logic formula becomes true. While 3-SAT is known to be theoretically hard to solve (NP-complete), in practice, not every problem is equally difficult.

This project investigates an interesting phenomenon known as the "Phase Transition" where 3-SAT problems suddenly switch from being mostly Satisfiable to Mostly Unsatisfiable at a specific ratio of constraints.

In this report, we aim to experimentally verify this transition. We generate thousands of random logic puzzles and use a modern SAT solver to analyze two things:

1. **Satisfiability:** At what point do the problems become impossible to solve?
2. **Difficulty:** How much actual computational work is required to find the answer?

Our goal is to demonstrate that the hardest problems aren't the impossible ones, but the ones right on the edge of being solvable.

2 Methodology: Defining Hardness

A critical contribution of this study is the selection of an appropriate metric for the measurement of "hardness."

2.1 The Flaw of Brute Force and Time

Initially, we considered measuring difficulty using the theoretical time complexity of a Brute Force solver, given by:

$$T(n,m) = O(m \cdot 2^n) \tag{1}$$

However, this metric is fundamentally flawed for experimental analysis for two reasons:

1. **Hardware Noise:** Wall-clock time is heavily influenced by external factors such as CPU caching, OS background processes, and thermal effects. These introduce significant "noise" that affects the algorithm's behavior.

2. **The "Luck" Factor:** Brute force solvers rely on checking assignments in a fixed or random order. A solver might stumble upon a satisfying assignment in the first microsecond purely by chance (Best Case $O(1)$), or it might search the entire space (Worst Case $O(2^n)$). This variance makes "time" a measure of luck, not difficulty.

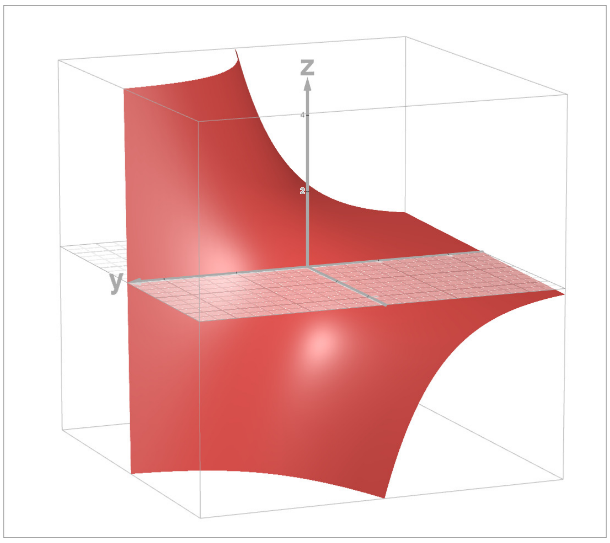

Figure 1: Theoretical Brute Force Complexity. It predicts a linear increase in difficulty as α increases, failing to predict the phase transition spike. (Z=Time Complexity Y=n X=m)

Furthermore, increasing the size of the expression (N) merely increases the "size of the domain." While this linearly increases the exponent for a Brute Force solver, it does not necessarily imply the problem structure is more complex; it simply means the haystack is larger.

2.2 The Systematic Approach: DPLL

To overcome the randomness of brute force, we utilize a **DPLL (Davis-Putnam-Logemann-Loveland)** based SAT solver (specifically *Glucose3*, a CDCL solver). Unlike brute force, DPLL approaches the problem systematically using:

- **Unit Propagation:** Forcing assignments that are logically implied.
- **Backtracking:** Systematically reversing decisions only when a contradiction is found.

This allows us to measure difficulty based on the **structure** of the constraints rather than the size of the search space.

Table 1: Comparison of Theoretical Prediction vs. Actual DPLL Performance

SAT Expression	**N**	**M**	**Exp. T(m,n)**	**Conflicts**	**DPLL Time**
$x_{97} \lor x_{78} \lor \neg x_{20} \land \dots$	100	200	$O(200 \cdot 2^{100})$	1	0.14 s
$x_{26} \lor x_{52} \lor \neg x_{13} \land \dots$	75	320	$O(320 \cdot 2^{75})$	53	0.82 s

Note: These DPLL times represent the average runtime over multiple seeded trials, minimizing the effect of hardware and OS noise.

2.3 The Invariant Metric: Conflicts

To isolate algorithmic difficulty from hardware variance, we adopted the number of **conflicts** (or deadends) as our primary metric.

A **conflict** represents a specific logical event: the solver navigates a branch of the search tree and encounters a contradiction, forcing a backtrack.

$$Hardness \propto \sum \text{Conflicts} \tag{2}$$

This metric is **invariant**. A deterministic solver running on a supercomputer will encounter the exact same number of conflicts as one running on a mobile phone for the same seed. It measures the "tangledness" of the logical structure, not the speed of the silicon.

3 Experimental Setup

Performing step wise analysis using Python and the `PySAT` library (wrapping the *Glucose3* solver).

- **Generation:** Uniform random 3-CNF formulas.
- **Variables (N):** Plot Hardness for every combination α, N for $N \in \{25, 50, 75\}$.
- **Sample Size:** 100 trials per α point to ensure statistical significance.
- **Resolution:** Use a fine-grained step size ($\Delta\alpha = 0.05$) near the critical region ($3.0 < \alpha < 6.0$) to capture the sharpness of the transition. $\Delta\alpha = 0.5$ otherwise.
- **Computation:** For every α, N combination keep track of conflicts for the entire sample size and finally calculate average conflicts, Probability of Satisfiability.
- **Probability of Satisfiability:** The number of clauses in our sample size that were Satisfiable.

4 The Phase Transition

The experiment yielded two important and correlated views of the transition.

4.1 Task 1: The Satisfiability Cliff

Figure 1 illustrates the probability that a random formula is satisfiable. For low α, $P(SAT) \approx 1$.

Intuition: Slicing the Search Space.

- To understand why low α problems are almost always solvable, consider the total search space of 2^N possible assignments.
- Geometrically, a single 3-SAT clause (e.g., $x_1 \vee x_2 \vee x_3$) acts as a filter that forbids exactly one specific combination of its variables [False, False, False] (ie forbids only 1/8th of the combinations).
- When α is low, the number of clauses M is small relative to N. The clauses alone only forbid a small number of solutions.
- Hence, a major chunk of the solutions are left valid.

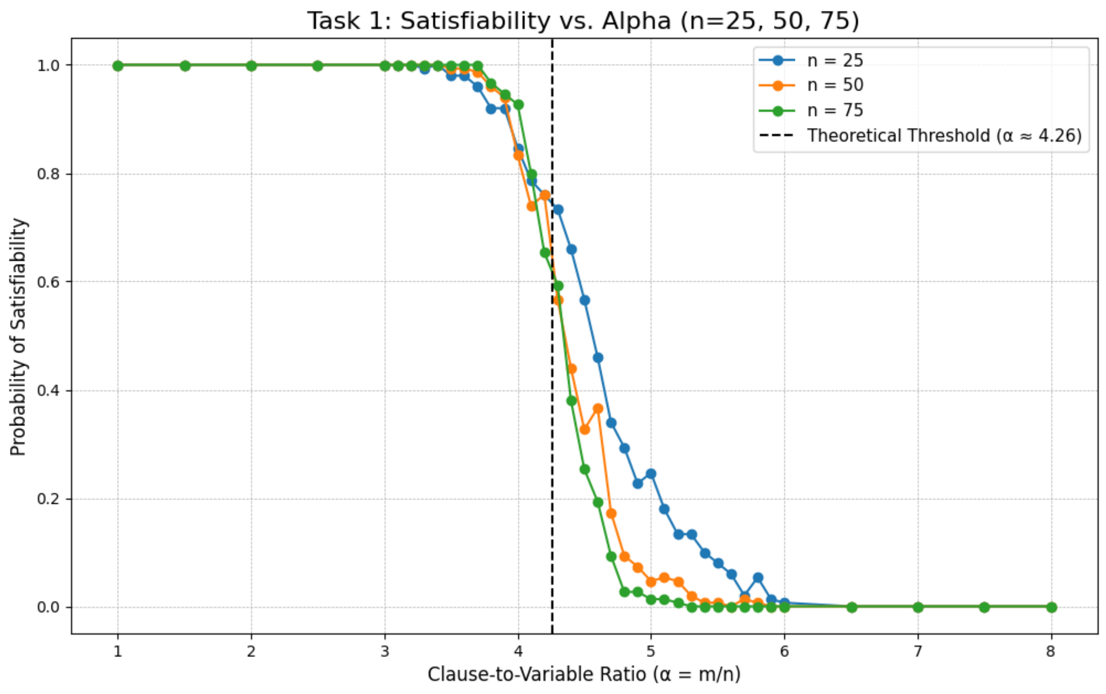

Figure 2: The "Cliff": Probability of Satisfiability vs. Ratio α for N=25, 50, 75. Note how the transition becomes sharper as N increases.

4.2 Task 2: The Difficulty Spike

Figure 2 plots the average number of conflicts. We observe an "Easy-Hard-Easy" pattern.

1. **Easy-SAT ($\alpha < 3$):** Solutions are dense; the solver finds one with minimal backtracking.

2. **Easy-UNSAT** ($\alpha > 6$)**:** The problem is over-constrained. Contradictions are shallow and easily found. UNSAT solutions are easy to find.

3. **Critical Region** ($\alpha \approx 4.26$)**:** The problem is critically constrained. Solutions are rare, and contradictions are deep, requiring exploration of a significant portion of the search tree.

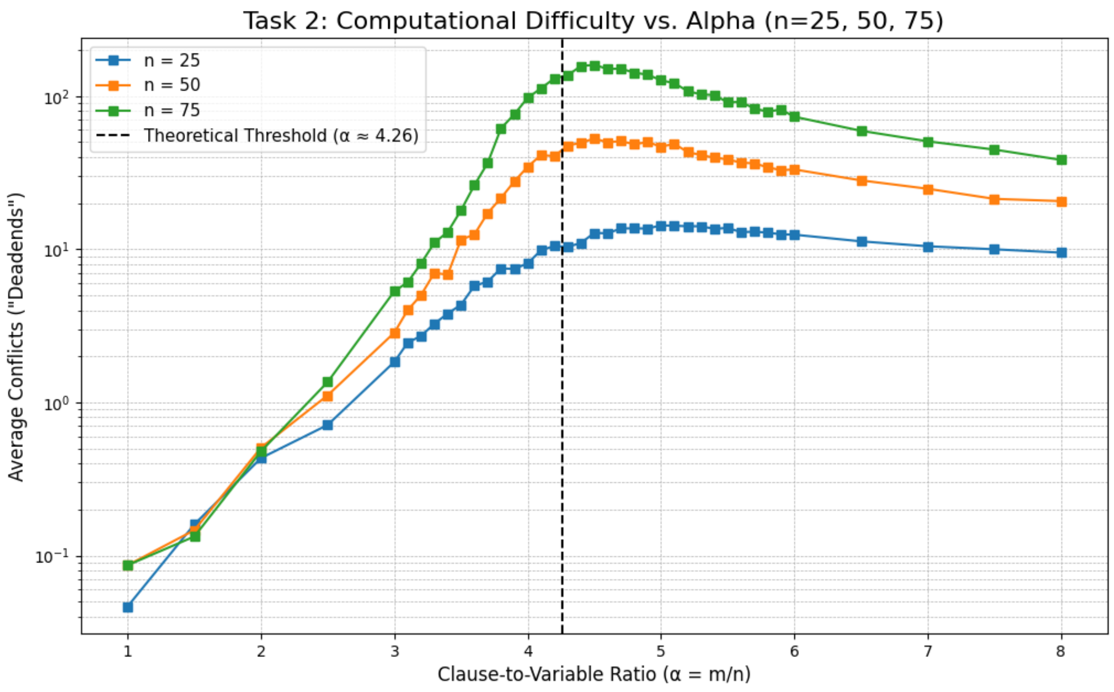

Figure 3: The "Spike": Computational Difficulty (Log Scale) vs. Ratio α. The difficulty peaks exactly at the phase transition.

4.3 The Combined View (N=75)

By overlaying both metrics (Figure 3), we confirm the hypothesis: the peak computational cost aligns perfectly(this is try to find :)) with the point of maximum uncertainty ($P(SAT) \approx 0.5$).

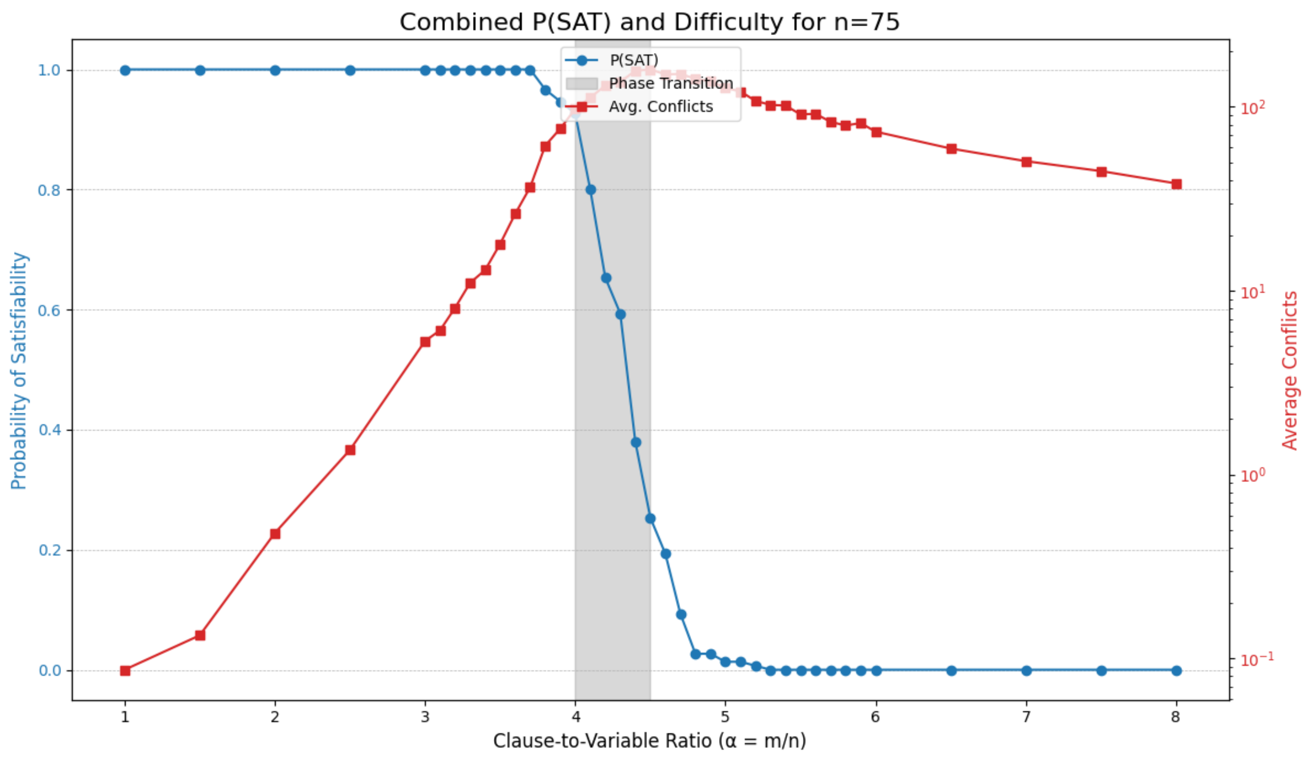

Figure 4: Correlation between Phase Transition and Difficulty for N=75.

5 Micro-Analysis: Computational Chaos

Standard benchmarks report average runtime, assuming a Normal distribution. However,analysis of the critical point ($\alpha = 4.25$) reveals a different kind of distribution. Isolating the raw conflict counts for 100 instances at the critical threshold and plotting them reveals that most of the instances have conflicts counts which are on the lower side, the average is pushed up heavily by the extreme cases.

5.1 Distributional Analysis: The Heavy Tail

Figure 5 presents the histogram of conflict counts. The data does not follow a standard Bell Curve; instead, it exhibits a **Heavy Tail**.

Formally, a heavy-tailed distribution is one where the probability of extreme values decays slower than exponentially (e.g., a Power Law). Unlike a Normal distribution, where extreme outliers are virtually impossible, a heavy tail implies that extreme events are rare but statistically expected. This confirms that algorithmic runtime is unstable: while the median run is fast, there is a significant risk of hitting an outlier that is orders of magnitude harder.

Mathematical Intuition. This instability arises from the mechanics of the DPLL search tree:

- **The Search Space:** The solver explores a binary tree of assignments.

- **Early Wrong Turns:** Occasionally, the solver makes an incorrect guess early in the process (at a shallow depth) that does not lead to an immediate contradiction.
- **Exponential Penalty:** The solver is forced to explore a deep "bad" subtree to prove the guess was wrong. Since tree size grows as 2^d (where d is depth), these specific wrong turns incur a massive, exponential cost compared to a typical run, creating the "heavy tail" (ie very high conflicts) in the data.

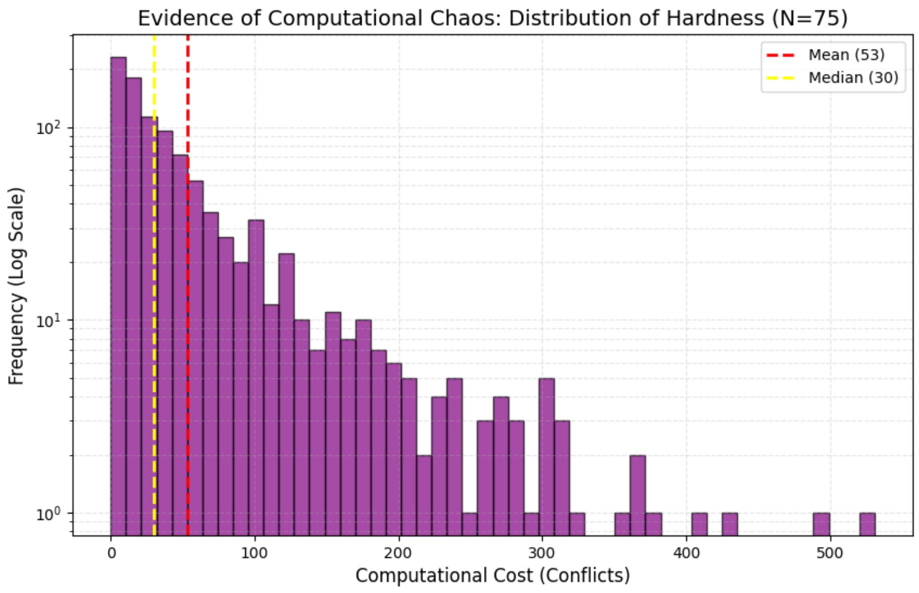

Figure 5: Histogram of conflicts at the ($\alpha = 4.1$). The long tail to the right indicates extreme volatility. The outliers push the mean up compared to the median as seen here

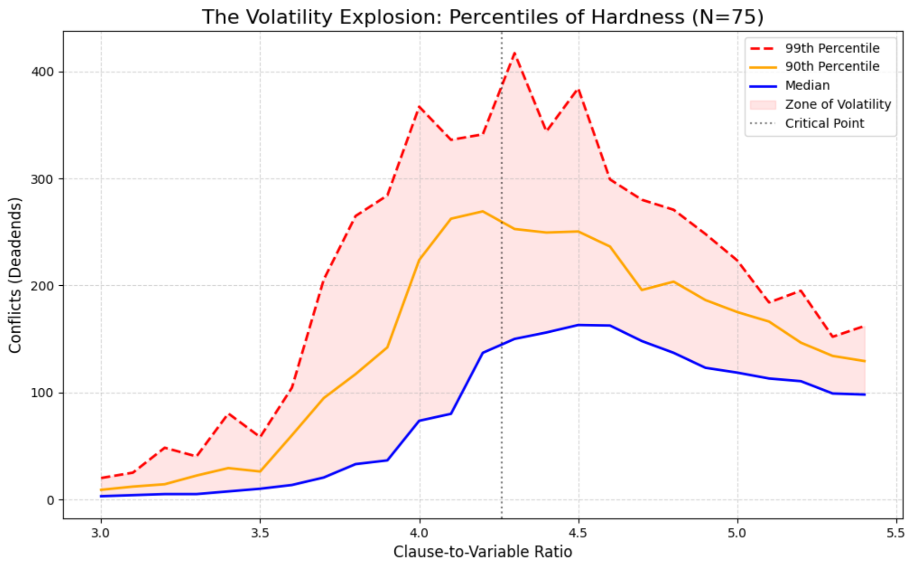

Figure 6: The Volatility Explosion. The shaded red region highlights the massive gap between the Median run and an extreme outlier (99th Percentile) at the critical threshold.

6 Implementation

Libraries used: PySAT, Matplotlib, numpy , pandas The python scripts used for analysis as well as the plots can be found on the Github repo.

7 Conclusion

We have successfully demonstrated that Random 3-SAT exhibits a sharp phase transition at $\alpha \approx 4.26$. By rejecting time complexity in favor of conflict analysis, we showed that the hardest instances are not those that are impossible (Easy UNSAT), but those that are barely possible. Furthermore, our distributional analysis reveals that the "critical point" is a region of high volatility, where "heavy tail" outlier instances are a large portion of the total computational effort.