Phase Transitions in 3-SAT

Chiara Peppicelli
chiara.peppicelli@edu.unifi.it
Mat: 7146573

Mirko Bicchierai
mirko.bicchierai@edu.unifi.it
Mat: 7146732

Jay Senoner
jay.senoner@edu.unifi.it
Mat: 7152874

Abstract

This paper presents an in-depth study of phase transitions in the 3-SAT problem, a canonical NP-complete problem in theoretical computer science and combinatorial optimization. The 3-SAT problem, which involves determining the satisfiability of Boolean formulas in conjunctive normal form with exactly three literals per clause, exhibits a critical phase transition as a function of the clause-to-variable ratio. Our research explores this phenomenon through empirical methods, revealing the intricate behaviors that emerge near the critical threshold. We employ large-scale computational experiments to empirically observe the sharp transition from satisfiable to unsatisfiable instances, in analogy of what is typical done when studying complex systems in statistical mechanics. Additionally, we investigate the role of different heuristic and exact algorithms in solving 3-SAT instances near the phase transition point. A Python implementation of all the described experiments can be found at `https://github.com/MirkoBicchierai/Statistical-Physics-SAT-Solver`

1. Introduction

The Boolean satisfiability problem (SAT) is a fundamental problem in computer science, specifically in the fields of logic, theoretical computer science, and artificial intelligence. It involves determining whether there exists an assignment of truth values to variables that makes a given Boolean formula true. The formula is typically expressed in conjunctive normal form (CNF). In CNF *literals* are grouped together to form *clauses*, which are assembled into a formula. By definition, a literal is a boolean variable in either affirmative or negative form, and a clause is a set of literals joined by the `OR` operation. A formula is said to be in CNF if it is expressed by a set of clauses joined by the `AND` operation.

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \tag{1}$$

The formula expressed in (1) is indeed in CNF, because it consists of a conjunction (`AND` operation) of clauses, each of which is a disjunction (`OR` operation) of literals. It is important to note that it has been proven that any Boolean expression can be converted into a semantically equivalent CNF formula, so there is no loss of generality in focusing on this one kind of expression. Using the (CNF) representation is highly advantageous for solving SAT instances. In the solving process, if a single clause is determined to be unsatisfiable, the entire formula is unsatisfiable since all clauses are connected by the `AND` operation. Additionally, if any literal within a clause is assigned a true value, the entire clause is considered true because all literals within a clause are connected by the `OR` operation.

2. The 3-SAT problem

Among the variants of the SAT problem, the 3-SAT problem is particularly notable. In the 3-SAT problem, each clause in the CNF formula contains exactly three literals. The problem can be formally defined as follows: given a Boolean formula in CNF with each clause containing exactly three literals, is there an assignment of truth values to the variables that satisfies the entire formula?

The significance of the 3-SAT problem lies in its status as an NP-complete problem, meaning that it is at least as hard as any problem in the class NP (nondeterministic polynomial time). This was demonstrated by Stephen Cook in his 1971 paper, where he proved that the general SAT problem is NP-complete, and Richard Karp's subsequent work, which included 3-SAT in his list of 21 NP-complete problems.

3. Phase Transitions

The 3-SAT problem is a pivotal benchmark in the study of computational complexity and algorithm performance due to its NP-complete nature. It is important to note that all complexity analyses, which categorize problems into distinct classes, are based on worst-case scenarios. Therefore, examining the distribution of hard and easy instances across the problem space is of significant interest. Empirical evidence has demonstrated that the ratio $\frac{M}{N}$ of clauses to variables is a critical parameter in characterizing the distribution of easy and hard 3-SAT instances. Specifically, it has been observed that as the ratio $\frac{M}{N}$ increases, the proportion of satisfiable 3-SAT instances decreases. Formulas with a low clauses-to-variables ratio can almost always be satisfied, since most of the variables appear only once or twice, and a conflict between them is unlikely. Conversely, when the clauses-to-variables ratio exceeds a certain threshold for a given Boolean formula, the formula is typically unsatisfiable. This is because, with significantly more clauses than variables, each variable is likely to appear in numerous clauses, leading to frequent conflicts. It turns out that there is a specific range of the clauses-to-variables ratio in 3-SAT instances that induces a phase transition. Within this range, the majority of instances abruptly shift from being satisfiable to unsatisfiable. In the subsequent sections, we aim to empirically investigate this phenomenon by implementing a SAT-solver algorithm and conducting various tests using randomly generated 3-SAT instances.

4. Experiments

In this section, we outline the experiments conducted to examine the distribution behavior of 3-SAT instances both near and far from the phase transition region. To solve the various instances, we implemented a backtracking based SAT solver that employs different heuristics to speed up the solving process. The first experiment, outlined in section 4.4, investigates the satisfability of distinct 3-SAT instances when the ratio of clauses-to-variables is gradually increased. This experiment aims to show the critical region of the clauses-to-variables ratio where the problem undergoes a phase transition. The second experiment, outlined in section 4.5, seeks to delve deeper into the behavior of this phase transition. It involves solving a variety of instances and analyzing how the percentage of satisfiable instances changes for a given number of variables in relation to the clauses-to-variables ratio. This analysis reveals that as the number of variables increases, the curve depicting the phase transition becomes steeper, eventually resembling a step function when the number of variables is sufficiently large. The third experiment, detailed in section 4.6, replicates the second experiment with a different focus. Here, our objective is to investigate how the mean computational cost of solving various instances for a fixed number of variables varies in relation to the clauses-to-variables ratio. In Sections 4.1, 4.2, and 4.3, we detail all the functions required to perform the outlined experiments, implemented in the Python language. All the outlined experiments refer to the previous work of Brian Hayes [2].

4.1. Generation Process

To conduct all the experiments detailed in section 4, the first step is to set up a process capable of generating different 3-SAT instances in conjunctive normal form (1).

Algorithm 1 3-SAT instance generation

```
def generate_cnf(nv, nc):
    cnf = []
    for _ in range(nc):
        clause_size = 3
        clause = set()
        literals = []
        while len(clause) < clause_size:
            while True:
                literal = random.randint(1,
                     nv)
                if not (literal in literals
                    ):
                    literals.append(literal
                        )
                    break
            if random.choice([True, False])
                :
                literal = -literal
            clause.add(literal)
        cnf.append(list(clause))
    return cnf
```

To generate a clause in a random 3-SAT instance, we choose three distinct variables from the total set of N variables. Each chosen variable is then either negated or left affirmative with a probability of $\frac{1}{2}$. To construct a 3-SAT formula with M clauses, we repeat this process M times. This repetition of the generation of a single 3-SAT instance allows us to setup all the experiments that require multiple instances for a given number of variables.

4.2. SAT Solver

The next step necessary to perform all the outlined experiments is to implement a backtracking-based SAT solver to solve all the generated instances. The solving process (2) performs an exhaustive search in the space of all possible variable assignments and then backtracks to a previous assignment when it is found that the current assignment leads to unsatisfiability.

Algorithm 2 Backtracking

```
def backtrack(formula, variables,
    labeling):
    if not formula:
        return labeling
    if any(not clause for clause in
        formula):
        return None
    variable = variables[0]
    remaining_variables = variables[1:]
    new_labeling = labeling.copy()
    new_labeling[variable] = True
    simplified_formula =
        simplify_formula(formula,
        variable, True)
    if simplified_formula is not None:
        result = backtrack(
            simplified_formula,
            remaining_variables,
            new_labeling)
        if result is not None:
            return result
    new_labeling[variable] = False
    simplified_formula =
        simplify_formula(formula,
        variable, False)
    if simplified_formula is not None:
        result = backtrack(
            simplified_formula,
            remaining_variables,
            new_labeling)
        if result is not None:
            return result
    return None
```

As we will see in the following sections, the time required to solve all the instances needed to conduct the outlined experiments using the described backtracking procedure becomes exponential near the phase transition. However, even when far from the phase transition region, the solving process remains considerably slow. The implementation of heuristics cannot solve the exponential time issue when near the critical region, however, it can significantly accelerate the process, even when inside the phase transition region.

The backtracking procedure (2) uses another function to perform a simplification (3) of the formula at a given step, when the variables are set to a given assignment. Since all the instances are in CNF form, if a single variable in a clause is set to

True, then that clause can be eliminated from the formula. Conversely, when a variable in a clause is set to False, that variable can be removed from the clause.

Algorithm 3 Simplify formula

```
def simplify_formula(formula, variable,
    value):
    new_formula = []
    for clause in formula:
        if value:
            if variable in clause:
                continue
            new_clause = [lit for lit in
                clause if lit != -variable]
        else:
            if -variable in clause:
                continue
            new_clause = [lit for lit in
                clause if lit != variable]
        if not new_clause and new_clause !=
            clause:
            return None
        new_formula.append(new_clause)

    return new_formula
```

4.3. Heuristics

An heuristic is a problem-solving approach or technique that employs practical methods, rules of thumb, or educated guesses to find solutions more quickly and efficiently, especially in situations where finding an optimal solution is impractical or impossible. To speed up the solving process, we implemented the unit propagation heuristic.

If a clause is a unit clause, i.e. it contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true. Thus, no choice is necessary. Unit propagation consists in removing every clause containing a unit clause's literal and in discarding the complement of a unit clause's literal from every clause containing that complement. In practice, this often leads to deterministic cascades of units, thus avoiding a large part of the naive search space.

Algorithm 4 Unit Propagation

```
unit_clauses = [clause[0] for clause in
    formula if len(clause) == 1]
for unit in unit_clauses:
    if debug:
        print(f"Unit propagation with unit
            clause: {unit}")
    if unit > 0:
        formula = simplify_formula(formula,
            unit, True)
        labeling[unit] = True
    else:
        formula = simplify_formula(formula,
            -unit, False)
        labeling[-unit] = False
    if formula is None:
        return None
    if abs(unit) in variables:
        variables.remove(abs(unit))

if not variables:
    return labeling
```

4.4. First experiment: Satisfiability

In this first experiment each of the 4.500 dots represents a single instance of the problem, figure 1 and 2; blue dots are satisfiable instances and red dots unsatisfiable. Height on the graph indicates the cost of finding a solution in terms of execution time. The cost reaches a peak where the instances change from mostly satisfiable to mostly unsatisfiable.

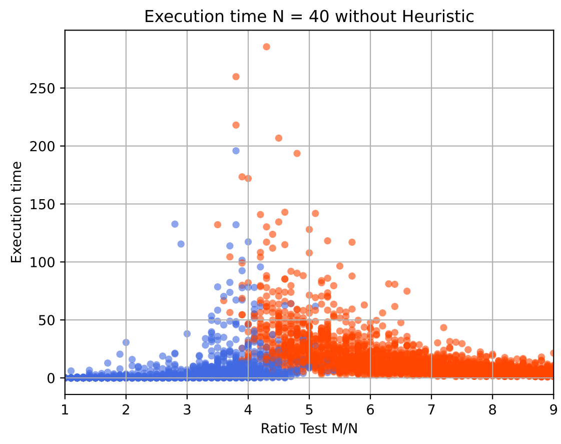

Figure 1. Satisfiability problem, SAT without heuristic for N = 40, undergoes a phase transition.

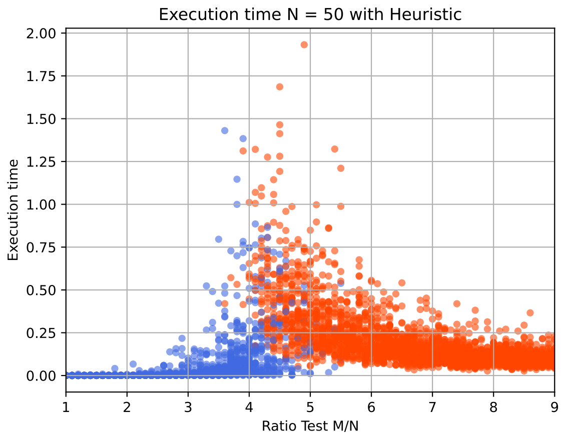

Figure 2. Satisfiability problem, SAT with heuristic for N = 50, undergoes a phase transition.

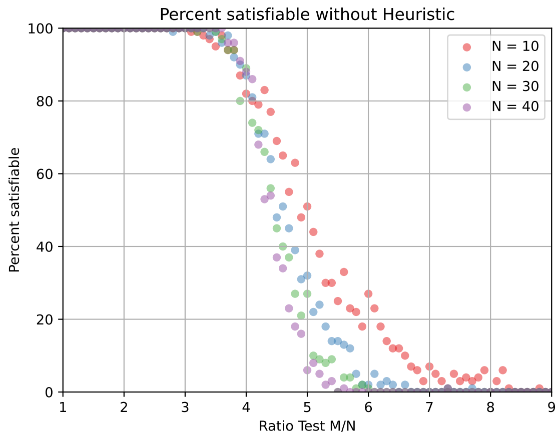

Figure 3. Transition from satisfiable to unsatisfiable gets steeper as the number of variables increases. Each dot is the average of 300 istances (without heuristics).

From the results of the first experiment, it can be observed that when the values of the clauses-to-variables ratio range approximately from 4 to 5, the problem undergoes a phase transition, where the majority of instances become unsatisfiable. Empirically, it is evident that the solver equipped with the unit propagation heuristic can solve all the 3-SAT instances much faster than the solver without the heuristic, even when the number of variables in the tested instances is higher for the solver with the heuristic.

4.5. Second experiment: Percentege of satisfiability

The objective of the second experiment is to study how the percentage of satisfiable instances varies in relation with the clauses-to-variables ratio. For smaller formulae, typically with $N = 10$ variables, the transition is relatively gradual. However, as the number of variables (N) increases, the transition becomes more pronounced. At $N = 50$, the likelihood of satisfiability remains high for M/N ratios up to approximately 4 (Figure 3, 4) . Beyond this threshold, the likelihood sharply declines, hovering close to zero for ratios exceeding 5. The abruptness of this transition presents an intriguing phenomenon, which intensifies as N approaches infinity, resembling a step function.

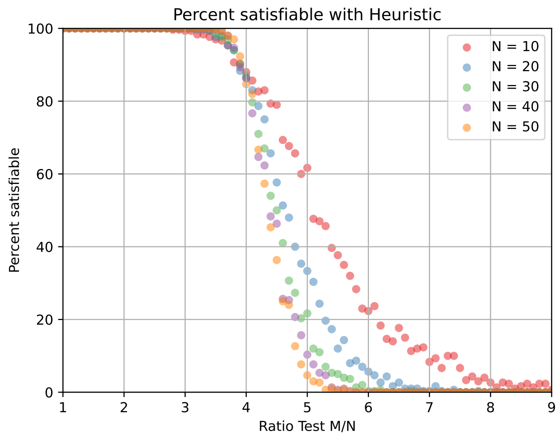

Figure 4. Transition from satisfiable to unsatisfiable gets steeper as the number of variables increases. Each dot is the average of 300 istances (with heuristics).

When testing the solver without heuristic, we limited the maximum number of variables in single instances to $N = 40$ instead of $N = 50$, in order to run the algorithm within our computational capabilities.

4.6. Third experiment: Execution time

In the third experiment, the main focus is to show how the mean execution time, that has been used as a metric for the computational cost required to run the experiment, evolves as the ratio of clauses-to-variables increases. This experiment runs on the same generated instances as the second experiment, in order to compare the different outputs. In scenarios where the ratio of clauses to variables is low, the problems tend to be relatively straightforward. Similarly, at very high ratios, the computational effort per problem only slightly increases. However, there exists a distinct peak in the difficulty curve within this range, where the average difficulty significantly rises; this peak aligns with the crossover region depicted in the probability graph. For any given value of N, the most challenging problems are concentrated around an M/N ratio close to the point where 50 percent of the formulas are satisfiable. Furthermore, as N grows and the crossover becomes more abrupt, the peak in the difficulty curve experiences a dramatic increase in magnitude. As we can see in Figure 6 and 5 the execution time of the solver without heuristics is significantly higher compared to when heuristics are applied. It's worth noting that for low N values, if we were to zoom in on the graph, we would still observe the critical region. However, in the plots, the higher values achieved for $N = 50$ and $N = 40$ overshadow those for lower N values. It is important to note that to execute this experiment in parallel to the secondwithout heuristics can take a significant amount of time.

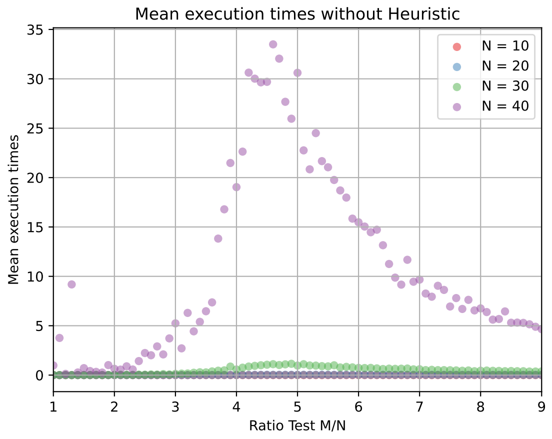

Figure 5. Peak in the cost, in terms of execution time, of finding solutions also gets sharper as the number of variable rises. Data are from the same istances as 3.

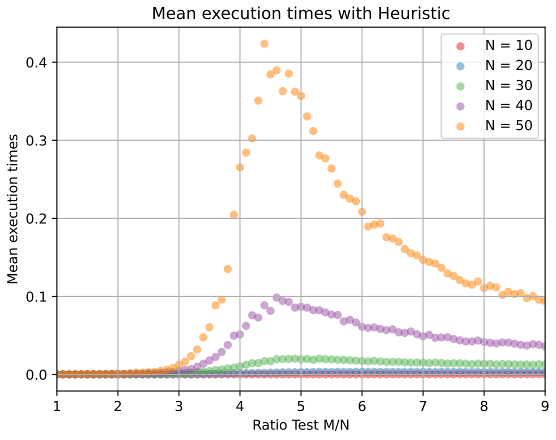

Figure 6. Peak in the cost, in terms of execution time, of finding solutions also gets sharper as the number of variable rises. Data are from the same istances as 4.

5. Conclusions

In this report, we analyzed the satisfiability of CNF boolean formulas as the ratio of the number of clauses to the number of variables varies within a certain range. To accomplish this, we implemented all the procedures required to solve a 3-SAT instance and then conducted various experiments to illustrate the critical region of values of the clauses-to-variables ratio that triggers a phase transition in the problem. The phenomenon of phase transition is analogous to observations in real physical systems, such as the behavior of a ferromagnet near the Curie point or the change of state of water with varying temperature from above to below zero.

The presence of a phase transition dependent on the variation of certain parameters can be regarded as a defining characteristic of all NP-complete problems, as previously observed in [1].

Since the SAT problem is NP-complete, only algorithms with exponential worst-case complexity are known for it. However, efficient and scalable algorithms for SAT have contributed to dramatic advances in our ability to automatically solve problem instances involving tens of thousands of variables and millions of constraints.

References

[1] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, page 331–337, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[2] B. Hayes. Can't get no satisfaction. 1997.

[3] T. Hogg, B. A. Huberman, and C. P. Williams. Phase transitions and the search problem. *Artif. Intell.*, 81:1–15, 1996.

[4] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. 07 1992.

[5] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. 2+p-sat: Relation of typical-case complexity to the nature of the phase transition, 1999.