Lab II
Phase transition in computational complexity: kSAT problem

Jakub Tworzydlo

Chair of Condensed Matter Theory
Institute of Theoretical Physics

3/03/2026 Pasteura, Warszawa

Plan

Plan

Intro

At the beginning of the 21st century, Mézard, Parisi, and Zecchina discovered that the methods of statistical physics could be highly useful in analyzing complex algorithmic problems. We will use a Python package with efficient kSAT solvers to reproduce some of their key results.

Specifically, we will investigate the phase transition in satisfiability (the probability that a random logical expression is satisfiable) for random 2SAT and 3SAT problems. This type of optimization problem is also naturally suited for D-Wave quantum annealers.

What is Pycosat?

`pycosat` is a Python binding to PicoSAT, a highly efficient Boolean satisfiability (SAT) solver written in C. It allows you to quickly solve complex logical formulas directly in Python by representing variables as non-zero integers and clauses as lists of integers. Minimalistic but sufficient documentation can be found at https://pypi.org/project/pycosat/.

Lab Computers
First, test from the Python interpreter to check if Pycosat is already installed:

```
>>> import pycosat
>>> pycosat.__version__
'0.6.6'
```

Personal Laptops
Open your terminal (Linux/macOS) or Command Prompt/PowerShell (Windows) and simply run:

```
pip install pycosat
```

Task 1: Pycosat Setup and Random kSAT

2 points Encode “by hand”, in plain Python (as a list of lists according to `pycosat` documentation) the following logical formula:

$$(p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (p \vee \neg q \vee \neg r)$$

Use `pycosat` to find a solution for p, q, r, which satisfy the formula and print them out.

Next, write a function that generates a random logical formula for $k = 3$ (3SAT) with N variables and M clauses.

- All variables within a single clause must be unique.
- Each variable should appear with a random sign (negation).

Hint: Check the documentation of `np.random.choice` and consider the option `replace=False` to draw variables without repetition.

Task 2: Satisfiability Transition and Complexity

3 points We will now observe the phase transition in the 3SAT problem and its impact on computational time.

For a fixed, large number of variables (e.g., $N = 50$), scan through different clause-to-variable ratios $f = M/N$. For each ratio, generate `nsamp` (e.g., 10 or 20) random samples.

Create a figure with two subplots:

1. **Satisfiability Ratio:** Plot the fraction of satisfiable expressions as a function of f.
2. **Computation Time:** Plot the average time it takes the solver to calculate the result as a function of f.

Narrow your calculation range to the vicinity of the transition (around $f \approx 4.2$). You should observe that the computation time drastically peaks exactly at the transition point. For a good plot you may need to increase N (at least $N = 200$).

Hint: Use the `time` module in Python (`time.time()`) to measure the execution duration of the solver.

Extra: Finite-Size Scaling and 2SAT

1 bonus point If you finish early, consider exploring the following extensions:

1. Finite-Size Scaling:
Repeat Task 2 for several different system sizes (e.g., $N = 20, 50, 100$). Plot the satisfiability ratio for all N on a single plot. How does the shape of the phase transition change as the thermodynamic limit ($N \to \infty$) is approached?

2. The 2SAT Problem (P vs NP):
Modify your random generator to produce 2SAT ($k = 2$) instances. Plot the satisfiability transition (which occurs at a different f). Is there a similar explosion in computation time at the transition? How does this reflect the fact that 2SAT is solvable in polynomial time, whereas 3SAT is NP-complete?