Lab III
Phase transition in computational complexity: kSAT problem

Jakub Tworzydło

Institute of Theoretical Physics
Jakub.Tworzydlo@fuw.edu.pl

15/03/2022 Pasteura, Warszawa

Plan

1. Intro

Plan

1. Intro

2. Tasks and hints

Intro

In the beginning of XXI century Mezard, Parisi and Zecchina discovered that the methods of statistical physics can be useful in analysis of complex algorithmic problems. We are going to use a Python package with efficient kSAT solvers to reproduce some of their results. We investigate a phase transition in satisfiability (ratio of satisfied logical expressions) for random 2SAT and 3SAT problems.

Pycosat installation and documentation

Linux

- run a command `pip install pycosat`
  (or `pip3` if you have a separate Python 3)
- same for Colab `!pip install pycosat`

Windows (? since I'm not an active user you may find a better way)

- download archive corresponding to your Python version and computer architecture
  https://www.lfd.uci.edu/~gohlke/pythonlibs/#pycosat
- for example for Python 3.6, 64-bit Windows
- run a command `pip install <file name>`

`Pycosat` is a simple wrapper for a proffesional SAT solver `Picosat`, which was a race-winning solution a decade ago. A minimalistic, but sufficient documentation you can find at https://pypi.org/project/pycosat/.

Linux from sources (lab computers)

first test from the interpreter if it is already installed

```
>>> import pycosat
>>> pycosat.__version__
u'0.6.3'
```

otherwise

- download the archive in the command line
  `wget  https://files.pythonhosted.org/packages/c0/fc`
- unpack `unzip pycosat-0.6.3.zip`
- add the path (run this command in home catalog)
  ```
  echo export PYTHONPATH="$PYTHONPATH:
  ~/lib/python3.8/site-packages/" >> ~/.bash_profile
  ```
- reset the system variables by running
  `source ~/.bash_profile`
- change directory `cd` to pycosat location and then install
  `python3 setup.py install --prefix=~`

Task 1

Encode "by had", in plain Python (as a list of lists according to `pycosat` documentation) the following logical formula

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

and print it. Use `pycosat` to find all solutions for p, q, r, which satisfy the formula and print them out.

Write a code, which prepares a random logical formula for a given k, M, N, where M denotes the number of clauses, N denotes the number of logical variables and k is the length of a single clause. All variables within a single clause have to be different, taken with a random sign (negation).

Test by printing a couple of examples for $k = 2$ and $k = 3$ and calculate their satisfiability. Hint: check the documentation of `np.random.choice` and consider the option `replace=False`.

Task 2

Generate `nsamp` random samples (logical expressions) and calculate the ratio of satisfiable expressions. Plot the satisfiability ratio as a function of $f = M/N$ for a fixed, big N (make sure that N denotes the number of variables).

Start testing with `nsampl=100` and smaller N, then increase N for quality plots. Narrow your the range of your calculation to the vicinity of the observed transition.

The final result should collect three values of N for $k = 2$ on a single plot, and similarly for $k = 3$ on another plot (or sub-panel).

What can one say about the character of transitions for $k = 2$ and $k = 3$?

Extra

Plot the average time of calculation versus $f = M/N$ close to the satisfiability transition. Take as big N as possible, compromise by taking fewer samples `nsampl=10`. Make sure that plots illustrate the transition both in time and in satisfability, and also that the difference between $k = 2$ and $k = 3$ cases is qualitatively noticable.

How would you summarize a qualitative difference between $k = 2$ and $k = 3$ case?