Structure and applications of boolean satisfiability problem

Michał Mrowczyk

Department of Computer Science
AGH Kraków

Pracownia Problemowa, 2015

Outline

1. Introduction And Motivation
2. My Work

SAT Decision Problem

- Given a boolean formula F decide (answer yes/no) whether there exists a satisfying assignment of true/false to variables (so that the formula evaluates to true).
- $x_1 \wedge (\overline{x_1} \vee x_2)$
- Setting $x_1 = 1$ and $x_2 = 1$ makes the formula above to evalutate to 1(true), so we call it satisfiable (SAT)
- What about this one: $(\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2)$?
- It turns out that this one always evaluates to 0 (false), so we call it unsatisfiable (UNSAT)

SAT Decision Problem

- Given a boolean formula F decide (answer yes/no) whether there exists a satisfying assignment of true/false to variables (so that the formula evaluates to true).
- $x_1 \wedge (\overline{x_1} \vee x_2)$
- Setting $x_1 = 1$ and $x_2 = 1$ makes the formula above to evalutate to 1(true), so we call it satisfiable (SAT)
- What about this one: $(\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2)$?
- It turns out that this one always evaluates to 0 (false), so we call it unsatisfiable (UNSAT)

Why is it an important problem to study ?

- Cook-Levin theorem: SAT is NP-complete
- All problems in NP can be encoded into SAT and solved using available SAT solvers
- Currently used SAT solvers can handle thousands and in some cases even millions of variables and constraints ! Solvers compete in annual competitions: `http://www.satcompetition.org/`

SAT Solvers

- Cryptominisat: `https://github.com/msoos/cryptominisat`
- PicoSAT: `http://fmv.jku.at/picosat/`
- Lingeling
- Glucose
- ...

Some Applications

- Planning
- Scheduling
- Bioinformatics e.g. protein folding
- Hardware and software verification
- FPGA routing
- ...
- Integer factorization - efficient polynomial time algorithm for SAT would even allow us to break RSA (However, it is highly unlikely that such algorithm exist...)

Plans

1. Generate various SAT instances by reduction from different problems (e.g. integer factorization)
2. Reduce some voting/election related problem (probably determining OWA-winner) to SAT
3. Try to understand the structure of SAT instances and their hardness for available solvers / heuristics

Work in progress - factorization

1. Problem of integer factorization using SAT:
    1. Input: integer n
    2. Output: boolean formula F for which the satisfying assigment is encoding factors (Formula F is UNSAT iff n is prime)
2. SAT instance (in DIMACS format) corresponding to the problem of factoring for number 42: `https://github.com/michal3141/sat/blob/master/data/42.dimacs`

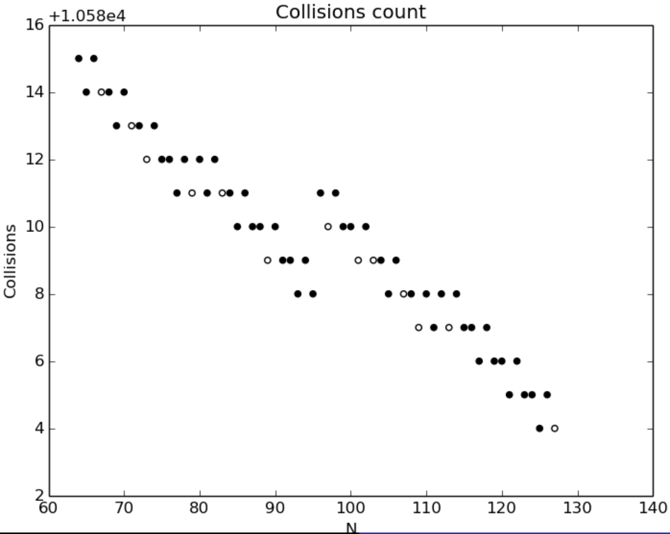
Factorization formulas - basic properties

1. Given two distinct *n*-bit integers, and boolean formulas generated for these integers, following holds true:

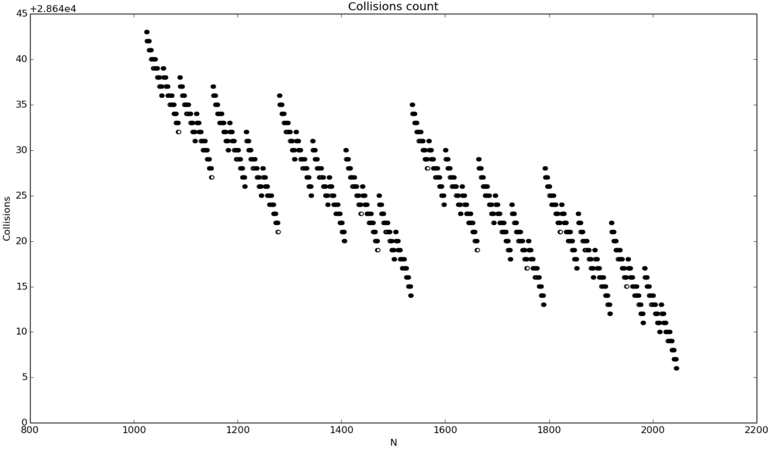
- Formulas have the same number of variables
- Formulas have the same number of clauses
- Formulas have the same number of literals
- Formulas have the same total number of negated and non-negated variables

2. From elementary number theory we know there is at least one prime number among n-bit integers (immediate consequence of Bertrand postulate [1])
3. It seems that when replacing ORs with XORs in formulas we are getting UNSAT instances (verified to few 1000's)

Number of variable collisions (7-bit)



Number of variable collisions (11-bit)



CP Modeling

- Modeling using various constraints and then translating models into SAT instances
- My approach to modeling: `https://github.com/michal3141/sat`
- Types of constraints I am currently using
  - inequalities, equalities
  - arithmetic operations (addition, multiplication, shift operations) - required for factorization
- (Much) More advanced modelling software/platforms:
  - MiniZinc: `http://www.minizinc.org/`
  - Numberjack: `http://numberjack.ucc.ie/`

Summary

- Currently implementing rudimentary CP modeling library so as to generate various types of SAT instances and ultimately understand hardness/structure of instances better
- TODO: Reducing OWA problem to SAT
- TODO: Evaluation of some algorithms (including classification heuristics) on generated SAT instances

For Further Reading I

Handbook of Satisfiability, Biere, A., Heule, M., Van Maaren, H., Walsh, T, February 2009

https://www3.nd.edu/~dgalvin1/pdf/bertrand.pdf

http://www.mimuw.edu.pl/~mati/fsat-20040420.pdf