Overview over SAT

Rohan Fossé

November, 6th. 2019

Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR-5800

Preliminaries

What is SAT?

Definition

Let $\Phi(a, b, ..)$ be a boolean formula.

Is there an **interpretation** of $(a, b, ..)$ that satisfies Φ ?

In theory

Cook-Levin's Theorem: SAT is **NP-Complete**.

$\Rightarrow$ First problem proved **NP-Complete**

$\Rightarrow$ P = NP?

In practice

- Many critical applications (e.g Configuration Management);
- Big size problems.

Notations

Literals

A literal $(a, b, \ldots)$ is either a boolean variable x or the negation of a boolean variable $\neg x$

Clauses

A clause C is a disjunction of literals *i.e*:

$C = a \vee b \vee .. \vee z$

Formula

A formula Φ is a conjunction of clauses *i.e*:

$\Phi = C_1 \wedge C_2 \wedge ... \wedge C_m$

Example

$\Phi = (a \vee \neg b) \wedge a \wedge (\neg a \vee \neg b)$

Resolution of a SAT formula

Let $\Phi_1 = (a \vee \neg b) \wedge a \wedge (\neg a \vee \neg b)$

Resolution

$$\begin{array}{ll} a & \neg b \\ a & \\ \neg a & \neg b \end{array}$$

Goal: Find an assignment of a and b such that each line is true ✓

a = ?

b = ?

Resolution of a SAT formula

Let $\Phi_1 = (a \vee \neg b) \wedge a \wedge (\neg a \vee \neg b)$

Resolution

a	¬b ✓
a	✓
¬a	¬b

Goal: Find an assignment of a and b such that each line is true ✓

a = `True`

b = ?

Resolution of a SAT formula

Let $\Phi_1 = (a \vee \neg b) \wedge a \wedge (\neg a \vee \neg b)$

Resolution

a	**¬b** ✓
a	✓
¬a	**¬b** ✓

Goal: Find an assignment of a and b such that each line is true ✓

a = `True`

b = `False`

Φ_1 is SAT ☺

Another resolution of a SAT formula

Let $\Phi_2 = (a \vee \neg b) \wedge b \wedge (\neg a \vee \neg b)$

Resolution

$$\begin{array}{ll} a & \neg b \\ b & \\ \neg a & \neg b \end{array}$$

Goal: Find an assignment of a and b such that each line is True ✓

a = ?

b = ?

Another resolution of a SAT formula

Let $\Phi_2 = (a \vee \neg b) \wedge b \wedge (\neg a \vee \neg b)$

Resolution

a	**¬b**
b	✓
¬a	**¬b**

Goal: Find an assignment of a and b such that each line is True ✓

a = ?

b = True

Another resolution of a SAT formula

Let $\Phi_2 = (a \vee \neg b) \wedge b \wedge (\neg a \vee \neg b)$

Resolution

a	**¬b x**
b	✓
¬a	**¬b x**

Goal: Find an assignment of a and b such that each line is True ✓

a = ☹

b = True

Φ_2 is UNSAT ☹

Polynomial-time Reduction

Definition

In computational complexity theory, a polynomial-time reduction is a method for solving one problem using another.

Example

- Reducing Graph Coloring to SAT
- Maximum Flow Problem to SAT

Why doing a reduction (in real life) ?

Mobile Radio Frequency Assignment

When frequencies are assigned to broadcast towers, if a tower is in the covering area of another one, they must not have the same assigned frequency in order to avoid wave interference. The goal is to minimize the number of used frequencies, since they are very expensive.

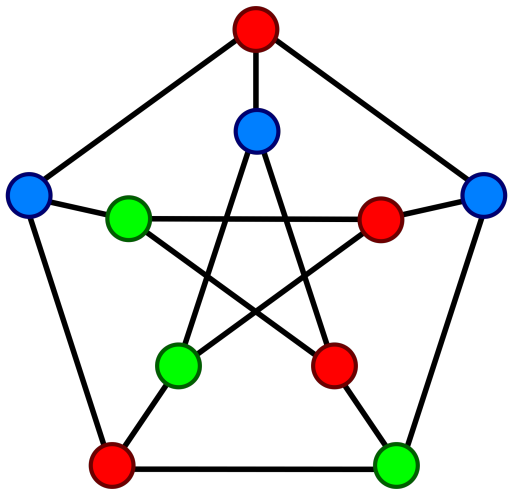

Figure 1: A proper vertex coloring of the Petersen graph

How to do this?

Reduce k-coloration to SAT

Goal: Reduce an instante of k-coloration of a graph $G(V, E)$ to an instance (formula) ϕ_G of SATCreate a boolean variable $x_{u,i}$ for each vertex u and each color i"For each edge uv, $color(u) \neq color(v)$" becomes $\bigwedge_{uv \in E} \bigwedge_{1 \leq i \leq k} (\neg x_{u,i} \vee \neg x_{v,i})$

Power of SAT solvers

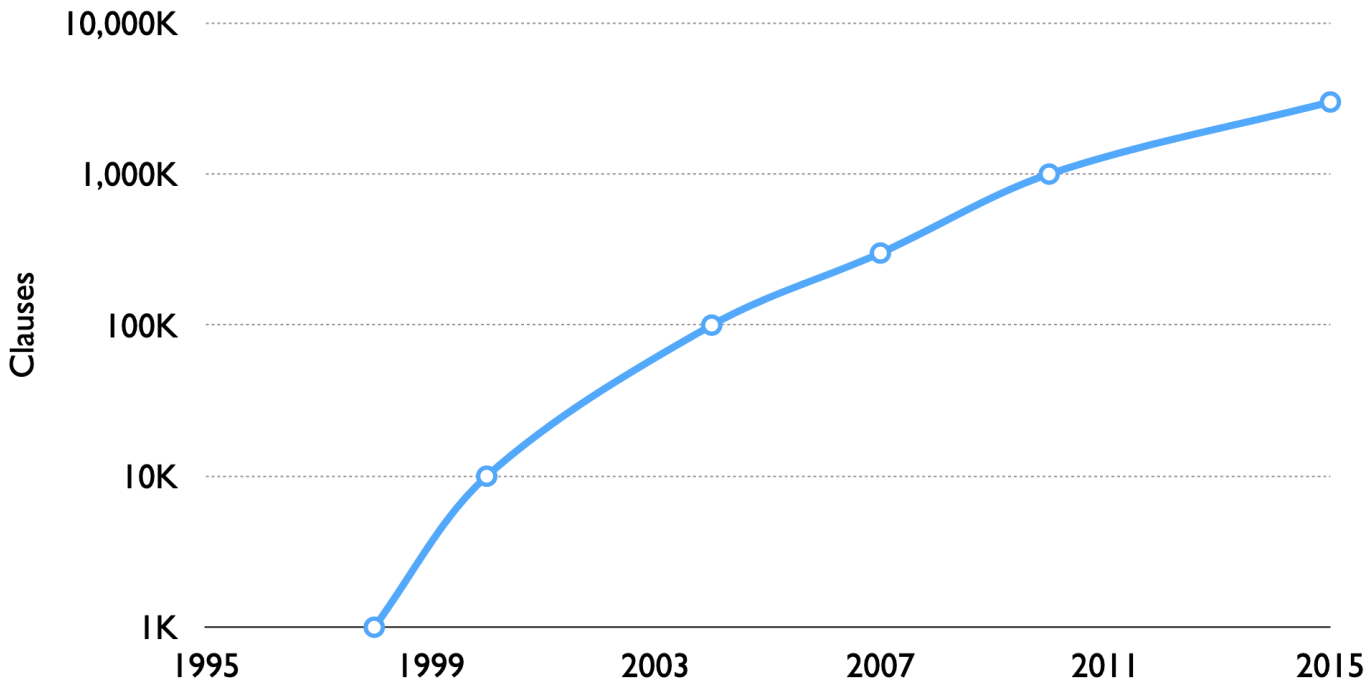

Figure 2: Evolution of SAT solver performance over time

SAT solvers

Pysat: A toolkit for SAT-based prototyping in Python

PySAT

PySAT is a Python (2.7, 3.4+) toolkit, which aims at providing a simple and unified interface to a number of state-of-art SAT solvers.

Pros:

- In Python;
- Great documentation;
- Easy to install (*pipinstallpython – SAT*)

(Major) Cons:

- Less powerful than other solvers

```
>>> from pysat.solvers import Glucose3
>>>
>>> g = Glucose3()
>>> g.add_clause([-1, 2])
>>> g.add_clause([-2, 3])
>>> print g.solve()
>>> print g.get_model()
...
True
[-1, -2, -3]
```

Figure 3: Trivial example using PySAT

Glucose

Glucose

`Glucose` is a winning award `SAT` solvers developped in `LaBRI` and `CRIL` by Laurent Simon and Gilles Audemard.

Pros:

- Developped in LaBRI ☺
- Powerful;
- Relatively easy to implement.

Cons:

- In C;
- It's hard to know which option to use;

Glucose

```
[% ./glucose ~/Desktop/These/Benchs-POS14/2008-satrace/satelited/een-pico-prop05-75.satelited.cnf.gz
c
c This is glucose 4.0 --  based on MiniSAT (Many thanks to MiniSAT team)
c
c ========================================[ Problem Statistics ]===========================================
c |                                                                                                       |
c |  Number of variables:         18188                                                                   |
c |  Number of clauses:           87504                                                                   |
c |  Parse time:                   0.05 s                                                                 |
c |                                                                                                       |
c | Preprocesing is fully done
c |  Eliminated clauses:           0.01 Mb                                                                |
c |  Simplification time:          0.05 s                                                                 |
c |                                                                                                       |
c ========================================[ MAGIC CONSTANTS ]==============================================
c | Constants are supposed to work well together :-)                                                      |
c | however, if you find better choices, please let us known...                                           |
c |-------------------------------------------------------------------------------------------------------|
c | Adapt dynamically the solver after 100000 conflicts (restarts, reduction strategies...)               |
c |-------------------------------------------------------------------------------------------------------|
c |                                |                                |                                     |
c | - Restarts:                    | - Reduce Clause DB:            | - Minimize Asserting:               |
c |   * LBD Queue    :     50      |   * First     :   2000         |    * size <  30                     |
c |   * Trail  Queue :   5000      |   * Inc       :    300         |    * lbd  <   6                     |
c |   * K            :   0.80      |   * Special   :   1000         |                                     |
c |   * R            :   1.40      |   * Protected :  (lbd)< 30     |                                     |
c |                                |                                |                                     |
c ================================[ Search Statistics (every  10000 conflicts) ]=========================
c |                                                                                                       |
c |          RESTARTS              |          ORIGINAL              |              LEARNT              | Progress |
c |       NB   Blocked  Avg Cfc |    Vars  Clauses Literals |   Red   Learnts    LBD2  Removed |          |
c ===============================================================================================================
c |       38         0      263 |   18131    87321   328060 |     2     5160     2076     4716 |  0.236 % |
c |       89        85      224 |   18119    87230   327836 |     3     8927     3811    10918 |  0.302 % |
c |      129       161      232 |   18113    87198   327772 |     3    18921     5193    10918 |  0.335 % |
c |      200       227      200 |   18111    87186   327748 |     4    19552     5975    20285 |  0.346 % |
c |      243       312      205 |   18108    87165   327694 |     4    29479     6770    20285 |  0.363 % |
c |      324       376      185 |   18108    87165   327694 |     5    26139     7200    33625 |  0.363 % |
c |      418       406      167 |   18105    87140   327621 |     5    35430     7494    33625 |  0.379 % |
c ===============================================================================================================
c restarts              : 451 (167 conflicts in avg)
c blocked restarts      : 419 (multiple: 149)
c last block at restart : 451
c nb ReduceDB           : 5
c nb removed Clauses    : 33625
c nb learnts DL2        : 7613
c nb learnts size 2     : 2311
c nb learnts size 1     : 72
c conflicts             : 75590          (28861 /sec)
c decisions             : 338134         (0.00 % random) (129101 /sec)
c propagations          : 22747444       (8685068 /sec)
c nb reduced Clauses    : 4211
c CPU time              : 2.61914 s

s UNSATISFIABLE
```

Sugar

Sugar

Sugar is a SAT-based Constraint Solver. Constraint Satisfaction Problem (CSP) is encoded to a Boolean CNF formula, and it is solved by an external SAT solver.

Why using CSP?

CSP represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods.

Real life example

automated planning, *lexical disambiguation*, *musicology* or *resource allocation*.

Conclusion

Thank you !

Questions ? ☺