# Satisfiability: Theory and applications

Mathieu Dutour Sikirić

Rudjer Bošković Institute, Croatia

September 29, 2022

# I. Introduction

# Boolean variables

- A Boolean variable is a variable that has two possible values, True or False.
- We have a number $N$ of Boolean variables $x_1, \ldots, x_N$.
- The negation of a Boolean variable $x$ is $\overline{x}$.
- A **clause** $c$ is a Boolean variable which is satisfied if

$$c = y_1 \wedge \cdots \wedge y_t \quad \text{with} \quad y_t = x_j \text{ or } \overline{x_j} \quad \text{for some} \quad j$$

  The clause is satisfied if at least one or more of the $y_i$ is satisfied.
- A satisfiability $S$ is a collection of clauses $c_1, \ldots, c_M$ such that

$$S = c_1 \vee \cdots \vee c_M$$

  A satisfiability is satisfied if all $c_i$ are True for some choice of the $x_i$. Otherwise it is **UNSAT**.

# Satisfiability problem

- **SAT**: The fundamental satisfiability problem is given a satisfiability problem written $N$ variables and $M$ clauses, to find some assignment of the $x_i$ which makes the satisfiability true.
- If no such assignment of Boolean variables exist, then the system is called unsatisfiable.
- Variant **MAXSAT**: allow some clauses to be true or false, but maximize the number of clauses being true.
- Variant **ENUMSAT**: enumerate all possible assignments of $x_i$ that makes the system true.
- Variant **Exactly-1 SAT**: Allow only one the variable to be true per clause.
- Many other variants, see `https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`

# II. Complexity Theory

# Satisfiability and complexity theory

- A *NP* ("Non-deterministic polynomial") problem means that it is possible to **test** that a **proposed** solution is indeed a solution in polynomial time.
- A *P* problem means that it possible to **find** a solution in polynomial time.
- Karp gave 21 combinatorial problems that are polynomially equivalent to **SAT** (which is *NP* and likely not *P*)
- Such problem are called *NP*-complete. There are now thousands of *NP*-complete problems.
- Example, solving linear systems $xA = b$:
  - A solution to the problem is a vector $x_0$ satisfying $x_0 A = b$ or a vector $w_0$ such that $Aw_0 = 0$ and $bw_0 \neq 0$.
  - The problem is in *NP*. If we have a solution, that is $x_0$ or $w_0$ it takes $O(N^2)$ polynomial time to check that it is indeed a solution.
  - The problem is in *P*. We have the Gauss method that allows to find solution in $O(N^3)$ time.
  - Explosion of the size of the coefficient is one aspect that can make things more complicated.

# The $P = NP$ problem

- The question $P = NP$ asks whether any $NP$ problem is actually also $P$.
- One example for it: The linear programming problem is a $NP$ problem that is also $P$.
- The millennium problem asks whether $P = NP$ (1 million dollar).
- If $P = NP$ then most cryptographic devices are broken, to check if a mathematical proof is correct is the same as writing it, or to be more poetic to appreciate great music is the same as writing it.
- In all likelihood, $P$ is not equal to $NP$.
- In practical terms, it means that solving satisfiability problem is not easy.

# Examples of NP-complete problems

- Graphs: Clique problem, Graph Coloring, Exact cover, Set packing, Subgraph isomorphism problem.
- Satisfiability with at most 3 clauses,
- Max Cut problem
- Subset sum problem
- Steiner tree problem
- Optimal solution for the $N \times N \times N$-Rubik cube.
- Video Games: Super Mario, Pokemon, Tetris, Candy Crush.
- See `https://en.wikipedia.org/wiki/List_of_NP-complete_problems`.

If for **any** one of those problem a polynomial time algorithm is found, then it is found for **all** $NP$-complete problems.

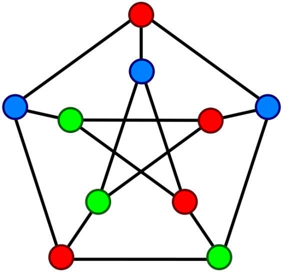
# How hard is SAT actually?

- If we have a satisfiability problem with $N$ variables and $M$ clauses actually there is better than a $2^N$ enumeration procedure, there are some algorithm with around $1.2^N$ steps needed.
- Still exponential in worst case most likely.
- But what about practical cases?
- The answer is that there are many different software for solving SAT problems: **minisat**, **glucose**, etc.
- There is a conference every year `http://www.satisfiability.org/` on the subject with tests and benchmarks and many categories:
  - Parallel track
  - Cloud track
  - Crypto track
  - Incremental Library track

# III. Application to Combinatorial problems

# Satisfiability for testing coloring

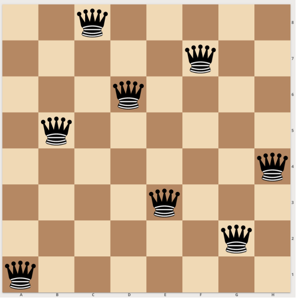
Given a graph on $n$ vertices, can it be colored with $c$ colors?



- We defined a number of Boolean $B_{v,i}$ with $v$ a vertex and $1 \leq i \leq c$ a color.
- We have following constraints:
  1. For vertex $v$ adjacent to $w$ we want for any $i$ to have $\overline{B_{v,i}} \wedge \overline{B_{w,i}}$
  2. For any vertex $v$ and colors $i < j$ we should have $\overline{B_{v,i}} \wedge \overline{B_{v,j}}$
  3. For any vertex $v$ we want $B_{v,1} \wedge B_{v,2} \wedge \cdots \wedge B_{v,c}$

# $N$ queens problems

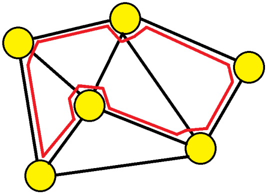
A classical problem we want to arrange $N$-queens so that none can attack another one. Example for $n = 8$:



- Define a variable $B_{i,j}$ for each entry of the square
- The constraints are:
  1. For each row $i$, at least one queen so $B_{i,1} \wedge B_{i,2} \wedge \cdots \wedge B_{i,N}$
  2. If $(i_1, j_1)$ and $(i_2, j_2)$ could attack each other then $\overline{B_{i_1,j_1}} \wedge \overline{B_{i_2,j_2}}$

# Hamiltonian paths

For a graph on $N$ vertices we want to find a path $v_1, \ldots, v_N$ passing by all vertices



- We write $B_{i,j}$ for the position $j$ in the $i$-th vertex of the path.
- We have following constraints:
    1. Only one position selected: $B_{i,1} \wedge B_{i,2} \wedge \cdots \wedge B_{i,N}$ and $\overline{B_{i,j}} \wedge \overline{B_{i,k}}$.
    2. If $i$ and $j$ are not adjacent then we set $\overline{B_{k,i}} \wedge \overline{B_{k+1,j}}$.

# Summary and experience

**Generalities**

1. We can use the work proving NP-completeness in order to relate a problem to the SAT.
2. There is a translation cost in term of encoding a problem into SAT. There can be several translations and some better than others.
3. What compensate is that the SAT solver are extremely well programmed with advanced optimization.

**The graph coloring problem**

1. There are some lower bound on the chromatic number computable in polynomial time from eigenvalues of the adjacency matrix.
2. For a graph with 16384 vertices, I could find a coloring with minisat in 2 minutes.
3. On the other hand computing the Hoffman lower bound was not possible in 2 hours.

# IV. Computer Games

# Sudoku game

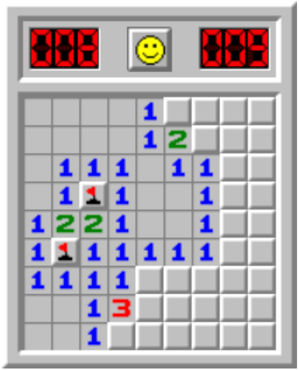
For squares, rows and columns only one value can occur:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

- Boolean variable $B_{i,j,k}$ for $1 \leq i, j, k \leq 9$
- Constraints.
  1. For already assigned entries $x(i,j)$ set a one clause $B_{i,j,x(i,j)}$ and $\overline{B_{i,j,k}}$ if $k \neq x(i,j)$.
  2. Always select one entry $B_{i,j,1} \wedge \cdots \wedge B_{i,j,9}$
  3. If two entries $(i_1, j_1)$ and $(i_2, j_2)$ are colliding we have $\overline{B_{i_1,j_1,k}} \wedge \overline{B_{i_2,j_2,k}}$ for all $k$.

# Minesweeper I

We have a partial solution



1. Variable $B_{ij}$ whether there is a mine or not.
2. For each entry $(i, j)$ the question becomes if adding $B_{ij}$ makes the problem feasible (in which case there could be a mine) or unfeasible (in which case there could not be a mine).

# Minesweeper II

1. We want to constraint that among 8 variables $x_1, \ldots, x_8$, exactly $k$ of them are true.
2. $POPCNT_0$: $\overline{x_1}, \ldots, \overline{x_8}$
3. $POPCNT_1$:
   3.1 $\overline{x_i} \wedge \overline{x_j}$ for $1 \leq i < j \leq 8$ and
   3.2 $x_1 \wedge \cdots \wedge x_8$
4. $POPCNT_k$:
   4.1 $\wedge_{i \in S} \overline{x_i}$ for all sets $S \subset \{1, \ldots, 8\}$ of size $k + 1$ and
   4.2 $\wedge_{i \in S} x_i$ for all sets $S \subset \{1, \ldots, 8\}$ of size $8 - k + 1$.
5. Putting together the $POPCNT_k$ we get the constraints for the minesweeper.
6. By iterating over the uncovered cells, looking for unsatisfiability, we can find the empty cells.
7. The above formulation is expensive, there are better ones.
8. See for details, **SAT/SMT by Example**, Dennis Yurichev.

# V. Scaling it up: Industrial applications

# Hardware verification

1. The Pentium division bug was a major problem discovered in 1994 that forced a recall of all processors:
   - Thomas Nicely, *Enumeration to* $10^{14}$ *of the twin primes and Brun's constant*, Virginia J. Sci. 46 (1995), no. 3, 195–204.
2. Pentium has just 3 million transistors while the i9 had about 7 billion transistors. So, why are they not recalled?
3. Part of the answer is that the CPU are tested by using satisfiability. See:
   - Per Bjesse, Tim Leonard, Abdel Mokkedem, *Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers*, International Conference on Computer Aided Verification (2001) 454–464

   Nowadays 70-80% of the expense of conceiving new electronic is in the verification.

# Algorithms for SAT

1. Main techniques:
   1.1 Conflict-Driven Clause Learning (CDCL) Solvers
   1.2 Variable Selection
   1.3 Literal Block Distance and Glue Clauses
   1.4 Stochastic Local Search (SLS) Solvers
2. It is accepted that all there is no universal technique for resolving SAT problems.
3. Machine learning techniques can be used to learn from partial information obtained in the computation.
4. Wenxuan Guo, Junchi Yan, Hui-Ling Zhen, Xijun Li, Mingxuan Yuan, Yaohui Jin, *Machine Learning Methods in Solving the Boolean Satisfiability Problem*

# Extensions

The success of SAT as a modeling tool has led to further extensions:

1. **Integer Programming**: Solving linear inequalities $f_i(x) \geq b_i$ for $x$ integer.
2. **Constraint Programming**: MiniZinc challenge `https://www.minizinc.org/challenge.html`
3. **Answer Set Programming**: It uses a format named **Lparse**.
4. **Satisfaction Modulo Theories** (SMT): The success of SAT is based on the simplest logic, Boolean variables. There are many other theories:
   4.1 bitvectors
   4.2 linear arithmetic, nonlinear arithmetic.

   The best solver is **Z3** and is used a lot in
   4.1 Formal verification of computer programs
   4.2 Automatic theorem proving

THANK YOU