SAT solver essentials, SAT modeling

Introduction

Gilles Audemard

VTSA School - Liege - 2021

Thanks to N. Szczepanski and L. Simon
Introduction

Summary

- Introduction
- CDCL solvers
- SAT incremental
- Parallel solvers
- Encodings

http://cours.alfweb.net
vtsa / vtsa
The SAT problem

- Variables: $x_1 \ldots x_3$, true or false
- Literals: $x_1$, $\overline{x_1}$ (or $\neg x_1$)
- Clauses: $\overline{x_1} \lor \overline{x_2} \lor x_3$
- CNF formula: conjunction of clauses

- SAT problem: is there an interpretation of variables that satisfy the CNF?
- The canonical NP-Complete problem
- The easiest of hard problems.
The SAT problem

- Variables: \( x_1 \ldots x_3 \), true or false
- Literals: \( x_1, \overline{x_1} \) (or \( \neg x_1 \))
- Clauses: \( \overline{x_1} \lor \overline{x_2} \lor x_3 \)
- CNF formula: conjunction of clauses

SAT problem: is there an interpretation of variables that satisfy the CNF?

The canonical NP-Complete problem

The easiest of hard problems.
The SAT problem

Variables: \( x_1 \ldots x_3 \), true or false

Literals: \( x_1 \), \( \overline{x_1} \) (or \( \neg x_1 \))

Clauses: \( \overline{x_1} \lor \overline{x_2} \lor x_3 \)

CNF formula: conjunction of clauses

SAT problem: is there an interpretation of variables that satisfy the CNF?

The canonical NP-Complete problem

The easiest of hard problems.
The SAT problem

- Variables: $x_1 \ldots x_3$, true or false
- Literals: $x_1, \overline{x_1}$ (or $\neg x_1$)
- Clauses: $\overline{x_1} \lor \overline{x_2} \lor x_3$
- CNF formula: conjunction of clauses

- SAT problem: is there an interpretation of variables that satisfy the CNF?
- The canonical NP-Complete problem
- The easiest of hard problems.
### About size of formula

<table>
<thead>
<tr>
<th>number of instructions</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^3 = 8$</td>
<td>immediate</td>
</tr>
<tr>
<td>$2^{37} \approx 80 \times 10^9$</td>
<td>1 second</td>
</tr>
<tr>
<td>$2^{56} \approx 8 \times 10^{16}$</td>
<td>$\approx 277$ hours</td>
</tr>
<tr>
<td>$2^{60} \approx 10^{18}$</td>
<td>166 days</td>
</tr>
<tr>
<td>$2^{128} \approx 340 \times 10^{38}$</td>
<td>$\geq 3$ billions of years</td>
</tr>
</tbody>
</table>

![Graph showing exponential growth](image-url)
SAT Solving in Practice

- Before 2000, reduce a problem to SAT to show that it is intractable
- After 2000, reduce a problem to SAT to solve it efficiently
- SAT is a success story in computer science
- Many practical applications

Source: J. Marques-Silva
Given a set \( \{1, 2 \ldots n\} \), it is possible to partition this set in two subsets, such that no part contains a triple \((a, b, c)\) such that \(a^2 + b^2 = c^2\).

For example, 3, 4 and 5 cannot be in the same subset, since \(3^2 + 4^2 = 5^2\).

The encoding is very simple:

- One variable per integer
- The variable is set to true if it is in one subset
- The variable is set to false in the other one

```
p cnf 7820 18930
  3 4 5 0
-3 -4 -5 0
  5 12 13 0
-5 -12 -13 0
  ...
```
Using a SAT solver (aka glucose)

- A solution exists for $n = 7824$
- and...
...There is no solution for $n = 7825$, the resulted SAT problem is unsatisfiable

A proof of size 200 terabytes!!, The largest math proof ever!!

Of course, need of proof checkers!!

Is it really a proof?

If mathematicians work is understood to be a quest to increase human understanding of mathematics, rather than to accumulate an ever-larger collection of facts, a solution that rests on theory seems superior to a computer ticking off possibilities. »

[Nature].

ticking off possibilities ??

Much more sophisticated than that (symmetry breaking, search of cubes...)

What about formal methods, SAT solving, Automated Theorem proving?
Why studying CDCL solvers

- The logic behind SAT is simple
- A CDCL solver is not too difficult to implement (a version with 500 loc is available with the presentation)
- Many open source solvers are available: Minisat is the most famous one
- You can easily modify them and try new techniques/ideas
- You need to think carefully with data-structures (many variables/clauses)

But... many promising ideas are finally ineffective

Playing with SAT solvers is addictive
From DP to CDCL

1960  Davis and Putnam algorithm : based on resolution rule : forget one variable after the other

Resolution rule

\[(x \lor y \lor \neg z) \otimes_x (\neg x \lor y \lor t) = y \lor \neg z \lor t\]
1960 Davis and Putnam algorithm: based on resolution rule: forget one variable after the other

Resolution rule

\[(x \lor y \lor \neg z) \otimes x (\neg x \lor y \lor t) = y \lor \neg z \lor t\]

\[x_1 \lor x_4\]
\[\neg x_1 \lor x_4 \lor x_{14}\]
\[\neg x_1 \lor \neg x_3 \lor \neg x_8\]
\[x_1 \lor x_8 \lor x_{12}\]
\[x_1 \lor x_5 \lor \neg x_9\]
\[x_2 \lor x_{11}\]
\[\neg x_3 \lor \neg x_7 \lor x_{13}\]
\[\neg x_3 \lor \neg x_7 \lor \neg x_{13} \lor x_9\]
\[x_8 \lor \neg x_7 \lor \neg x_9\]
1960  Davis and Putnam algorithm: based on resolution rule: forget one variable after the other

Resolution rule

\[(x \lor y \lor \neg z) \otimes_x (\neg x \lor y \lor t) = y \lor \neg z \lor t\]
1960  Davis and Putnam algorithm: based on resolution rule: forget one variable after the other

Resolution rule

\[(x \lor y \lor \neg z) \otimes_x (\neg x \lor y \lor t) = y \lor \neg z \lor t\]

- \(x_4 \lor x_{14}\)
- \(x_4 \lor \neg x_3 \lor \neg x_8\)
- \(x_8 \lor x_{12} \lor x_4 \lor x_{14}\)
- \(x_5 \lor \neg x_9 \lor x_4 \lor x_{14}\)
- \(x_5 \lor \neg x_9 \lor \neg x_3 \lor \neg x_8\)

- \(x_2 \lor x_{11}\)
- \(\neg x_3 \lor \neg x_7 \lor x_{13}\)
- \(\neg x_3 \lor \neg x_7 \lor \neg x_{13} \lor x_9\)
- \(x_8 \lor \neg x_7 \lor \neg x_9\)

- Combinatorial explosion
- Used in pre-processing
some comparisons... even then

« The superiority of the present procedure (i.e. DP) over those previously available is indicated in part by the fact that a formula on which Gilmore’s routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using the present method in 30 minutes »

[Davis et Putnam 1960], page 202.
From DP to CDCL

1960  Davis and Putnam algorithm: based on resolution, forget one variable after the other
1962  Davis, Logeman Loveland algorithm: backtrack search tree (partial models)

- Decision variable is important
- Mistakes at the top of the search are dramatic
- A perfect branching scheme is NP-Hard (if the formula is SAT, I can make find a solution without backtrack)
## From DP to CDCL

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>Davis and Putnam algorithm: based on resolution, forget one variable after the other</td>
</tr>
<tr>
<td>1962</td>
<td>Davis, Logeman Loveland algorithm: backtrack search tree (partial models)</td>
</tr>
<tr>
<td>1962 – 2001</td>
<td>Improve DPLL solvers: heuristics, look-ahead strategies...</td>
</tr>
</tbody>
</table>
## From DP to CDCL

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>Davis and Putnam algorithm (DP) : based on resolution, forget one variable after the other</td>
</tr>
<tr>
<td>1962</td>
<td>Davis, Logeman Loveland algorithm (DLL, DPLL) : backtrack search tree (partial models)</td>
</tr>
<tr>
<td>1962 – 2001</td>
<td>Improve DPLL solvers : heuristics, look-ahead strategies...</td>
</tr>
<tr>
<td>1992</td>
<td>”Planning as Satisfiability”. Kautz and Selman. ECAI</td>
</tr>
<tr>
<td>1999</td>
<td>”Symbolic Model Checking Without BDD”. Biere, Cimatti, Clarke and Zhu. TACAS</td>
</tr>
</tbody>
</table>

- Encoding of application problems in SAT
## From DP to CDCL

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960</td>
<td>Davis and Putnam algorithm (DP) : based on resolution, forget one variable after the other</td>
</tr>
<tr>
<td>1962</td>
<td>Davis, Logeman Loveland algorithm(DLL, DPLL) : backtrack search tree (partial models)</td>
</tr>
<tr>
<td>1962 – 2001</td>
<td>Improve DPLL solvers : heuristics, look-ahead strategies...</td>
</tr>
<tr>
<td>1992</td>
<td>&quot;Planning as Satisfiability&quot;. Kautz and Selman. ECAI</td>
</tr>
<tr>
<td>1999</td>
<td>&quot;Symbolic Model Checking Without BDD&quot;. Biere, Cimatti, Clarke and Zhu. TACAS</td>
</tr>
<tr>
<td>1996</td>
<td>&quot;GRASP - a new search algorithm for satisfiability&quot;. Sakallah and Marques-Silva. ICCAD</td>
</tr>
<tr>
<td>2001</td>
<td>&quot;Chaff : Engineering an Efficient SAT Solver&quot;. Moskewicz, Madigan, Zhao, Zhang and Malik. DAC</td>
</tr>
<tr>
<td>2003</td>
<td>&quot;An extensible SAT solver&quot;. Een and Sorensson. SAT</td>
</tr>
</tbody>
</table>

- Learning + dedicated data structures for managing huge formulas
Before 2000

- **Look Ahead solvers**
  - To know where to go, you need to know where you are
    - Search tree
    - Ideas/Techniques that aim to reduce/balance the search tree.
    - Heuristics, failed literals
    - Explanation of performances is (relatively) simple

- **Lookback solvers (CDCL)**
  - You do not know where you are, but you know where to go
    - Many variables, many clauses
    - Restarts
    - Explanation of performances is quite hard
Performances after 2001

2002
Performances after 2001
Performances after 2001

2005
Performances after 2001

2007
Performances after 2001
Performances after 2001

- 2011

Graph showing the number of solved problems (over the 300 benches from 2011) against the maximum allowed time (seconds).
Performances after 2001

![Graph showing performances after 2001]

The graph illustrates the number of solved problems (over the 300 benchmarks from 2011) against the maximum allowed time (seconds) for various years, with a focus on 2014.
Performances after 2001
Performances after 2001
Performances after 2001

2020
### The precursors
- Grasp (1996) Learning
- Relsat (1996) Learning
- Zchaff (2000) Lazy data structures

### Minisat based
- Minisat (2003), the original one
- SAT4J (2004), in Java
- RSat (2007), phase saving
- Glucose (2009), quality of learnt clauses
- Cryptominisat (2009), XOR reasoning
- MapleXXXX (2016...), based on glucose. Add LRB heuristic and other features

### Armin Biere solvers
- Precosat (2009)
- lingeling (2010)
- cadical (2019)
- Kissat (2020)
SAT Heritage

- [https://github.com/sat-heritage/docker-images](https://github.com/sat-heritage/docker-images)
- Software heritage for the SAT community
- A collection of SAT solvers that participate to some SAT competitions (632)
- Local search, parallel, CDCL, DPLL...
- You can search, download, build, launch them
- Based on Docker
Why studying CDCL solvers

- The logic behind SAT is simple
- A CDCL solver is not too difficult to implement (see https://github.com/audemard/minicdcl)
- Many open source solvers are available: Minisat is the most famous one
- You can easily modify them and try new techniques/ideas
- You need to think carefully with data-structures (many variables/clauses)

But.. many promising ideas are finally ineffective

Playing with SAT solvers is addictive
Exercise
Pythagorean triples
A word on 3-SAT

- $k$-SAT: Exactly $k$ literals per clause
- 2-SAT is polynomial
- 3-SAT is NP-Complete