The Boolean Satisfiability Problem: an overview of solving techniques and applications

Clémentin Tayou Djamegni

Professor, University of Dschang

May 19, 2021

Outline

1. Background
   - Definitions and preliminaries
   - SAT Problem
   - Theoretical and practical interest on SAT
2. How SAT Solvers work
   - Solving Algorithms (Sequential and parallel)
   - Some techniques used to boost the performance of solvers
   - Some state-of-the-art SAT solvers
   - Input format for SAT solvers and output
3. Applications of SAT
   - Schema for solving a problem using a SAT solver
   - CNF encoding of cardinality constraints for problem modeling
   - Example on the Graph coloring problem
   - Other applications

Outline

Decision problems

Decision problem

A decision problem is a problem which the answer is either YES or NO.

Some well-known classes of decisions problems

- P is the set of decision problems that can be solved in poly-time,
- NP is the set of decision problems that the YES answer can be verified in polynomial time (Certified YES answer is easy to verify),
- co-NP is the set of decision problems that the certified NO answer can be verified in polynomial time.

Refer to **(Ding-Zhu Du 2000)** for formal definitions of these classes.

Open questions

- Does P = NP ?
- Does NP = co-NP ?

Propositional Logic I

Propositional variable

Variable whose domain is $\{true, false\}$ also denoted $\{0, 1\}$ or $\{\bot, \top\}$.
Example : $x_1, \cdots, x_n$

Literal

It is a propositional variable (positive literal) or the negation of a propositional variable (negative literal).
Example : $\neg x_1, x_1, \neg x_2, x_2, \cdots$

Clause

Finite disjunction of literals. **Example :** $C = (x_1 \vee \neg x_2 \vee x_3)$

Propositional Logic II

CNF Formula

Finite conjunction of clauses. **Example :**
$\mathcal{F} = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$

Interpretation

An interpretation of a CNF formula $\mathcal{F}$ is an assignment of truth values to its variables.

Model

A model for a CNF formula $\mathcal{F}$ is an interpretation that makes $\mathcal{F}$ evaluates to *true*, i.e. an interpretation that set at least one literal of each clause of $\mathcal{F}$ to *true*.

Propositional Logic III

Resolution

The resolution rule is an inference rule that takes two clauses $C = \alpha \vee x$ and $C' = \beta \vee \neg x$ where α, β are disjunction of literals and infers the clause $C \bowtie_x C' = \alpha \vee \beta$. $C \bowtie_x C'$ is called the resolvent of C and C' on variable x.

Example

If $C = x_1 \vee \neg x_2$ and $C' = \neg x_1 \vee x_3$, then the resolvent of C and C' on x_1 is $C'' = \neg x_2 \vee x_3$

The resolvent is always logically implied by the clauses from which it is built. $C \wedge C' \models C''$.

Resolution graph

Consider the CNF formula $\mathcal{F} = \{a \vee \neg d \vee \neg c, c \vee \neg b, b \vee d\}$ and the clause $\alpha = a \vee c \vee \neg b$. The derivation of α from $\mathcal{F}$ is depicted by the following resolution graph :

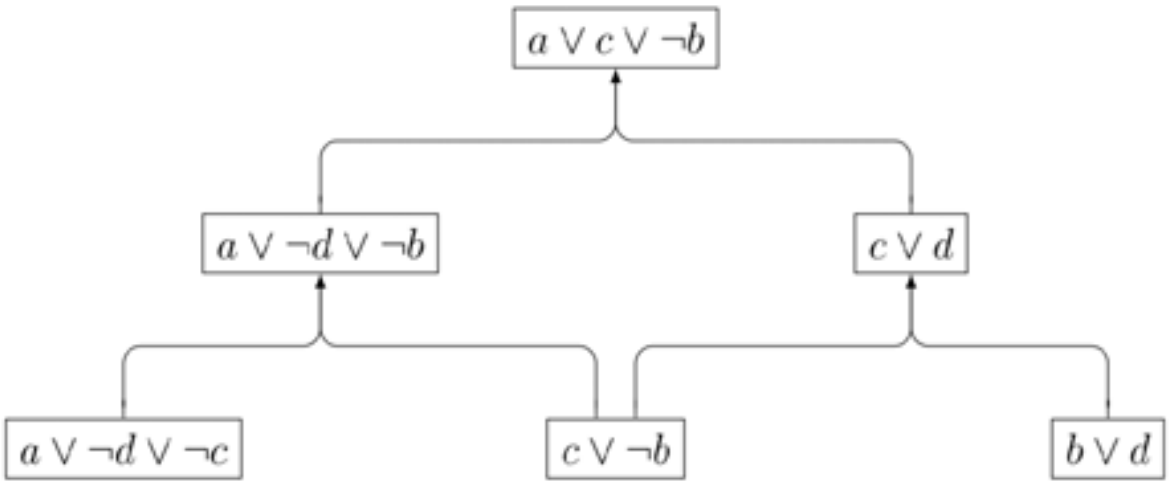


Figure – Resolution graph

Proof by resolution systems I

- **Regular resolution :** No path from a leaf to the root where a variable is eliminated more than once per resolution.
- **P-resolution and N-resolution :** A clause participating in the resolution contains only positive literals.
- **Extended resolution :** At each stage of the construction of the proof, it is possible to add lemmas to the formula, in the form of a new variable y associated with the clauses which encode $y \leftrightarrow (l_1 \vee l_2)$.
- **Linear resolution :** The principle consists in restricting the sequence of resolution in order to consider only linear derivations.
- **Unit resolution :** One of the clauses to be resolved is a unitary clause. Obviously this method is not complete in general, but it is complete for Horn clauses.

Proof by resolution systems II

- **Semantic resolution :** using a model or an interpretation to guide the search for proof by rebuttal.
- **Restricted resolution :** Add to the formula a number of clauses produced by the resolution and run a solver based on the regular resolution.

The Boolean Satisfiability Problem (SAT) I

Satisfiable and unsatisfiable CNF formula

A CNF formula is *satisfiable* if there is an assignment of truth values to its variables so that at least one literal of each clause is set to *true*. It is *unsatisfiable* otherwise.

Definition (SAT)

- Input : a propositional logic formula $\mathcal{F}$ (usually in CNF) ;
- Output : the answer to the question « is $\mathcal{F}$ satisfiable ? »

Example

- $\mathcal{F}_1 = (x_1 \vee x_2) \wedge (\neg x_1) \wedge (\neg x_1 \vee x_2)$ is satisfiable since setting $x_1 = \mathit{false}, x_2 = \mathit{true}$ makes $\mathcal{F}_1$ satisfied,
- $\mathcal{F}_2 = (x_1 \vee x_2) \wedge (\neg x_1) \wedge (x_1 \vee \neg x_2)$ is unsatisfiable.

The Boolean Satisfiability Problem (SAT) II

SAT solver

A SAT solver is a computer program built to solve SAT. SAT solvers usually provide certificates of their answers, that is, a truth assignment that satisfies the input formula in the case it is satisfiable or an unsatisfiability proof (typically in DRUP or DRAT formats) when the formula is unsatisfiable.

Cook Theorem **(Cook 1971)**

SAT is NP-complete.

What does it exactly mean ?

NP-Completeness of SAT

SAT cannot be solved in poly-time unless $P = NP$.

The Boolean Satisfiability Problem (SAT) III

k-SAT $k \geq 3$ is NP-complete (Deng and Xu 2008)

the k-SAT problem which consists of deciding the satisfiability of a k-CNF formula is NP-complete for $k \geq 3$.

(Krom 1967; Aspvall, Plass, and Tarjan 1979)

2-SAT is in P.

(Dowling and Gallier 1984)

HORN-SAT is in P.

Great interest for SAT

On the theoretical level

Reference NP-complete problem : SAT is the first problem to have been shown NP-complete

On the practical level

SAT is used for solving many other fields of computer science

International Competitions

Organization every year of SAT competitions (SAT Race, SAT Competition, SAT Challenge) at the end of which are awarded the best solvers.

What famous computer scientists said about SAT

Donald Knuth, 1974 ACM Turing Award Recipient

SAT is evidently a killer app, because it is key to the solution of so many other problems.

Stephen Cook, 1982 ACM Turing Award Recipient

The SAT problem is at the core of arguably the most fundamental question in computer science : What makes a problem hard ?

Edmund Clarke, 2007 ACM Turing Award Recipient

Clearly, efficient SAT solving is a key technology for 21st century computer science.

Outline

1. Background

2. How SAT Solvers work
   - Solving Algorithms (Sequential and parallel)
   - Some techniques used to boost the performance of solvers
   - Some state-of-the-art SAT solvers
   - Input format for SAT solvers and output

3. Applications of SAT

SAT solving algorithms I

SAT solving algorithms are broadly classified into two main classes :

- Complete algorithms
- Incomplete algorithms

1. Incomplete (most of them based on stochastic local search)
   - SAT is seen as an optimization problem where the goal is to minimize the number of unsatisfied clauses
   - Start with a complete random assignment
   - Repeatedly flip (randomly/heuristically chosen) variables to decrease the number of unsatisfied clauses
   - Introduce perturbations to avoid local minima
   - Local search algorithms are incomplete : they cannot prove unsatisfiability !

SAT solving algorithms II

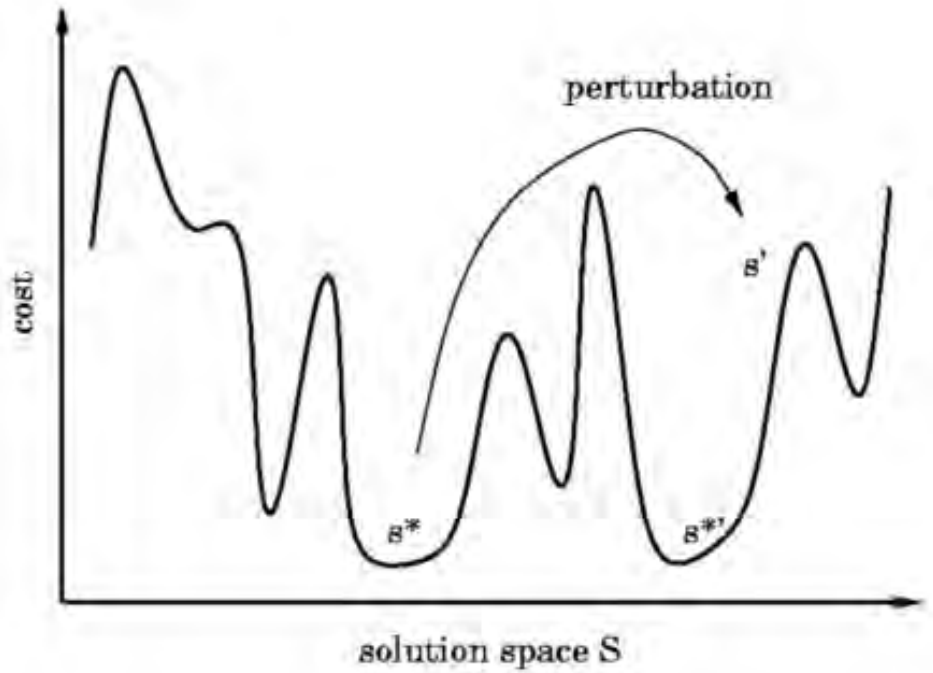


Figure – [Alan Mackworth, UBC, Canada] Illustration of Stochastic Local Search

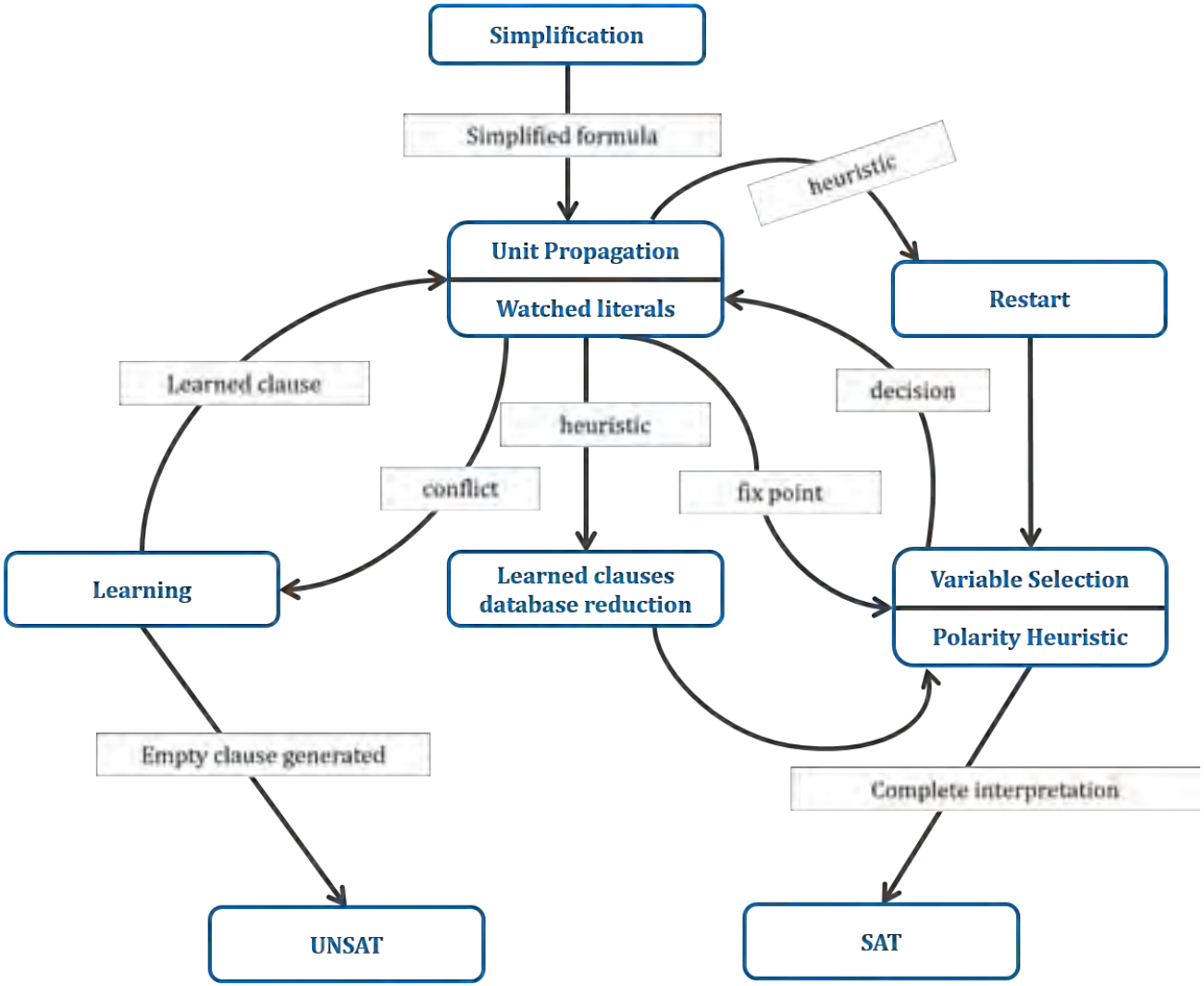
SAT solving algorithms III

2. Complete
   1. DP **(Davis and Putnam 1960)**
      - Iteratively select a variable x to perform resolution on,
      - Replace all clauses containing x and $\neg x$ by all possible resolvent on x that can be generated.
      - Termination : if either the empty clause was derived (conclude UNSAT) Or all variables have been selected (SAT)
   2. DPLL **(Davis, Logemann, and Loveland 1962)**
      - Choose a literal, assign a truth value to it and simplify the formula
      - Recursively check if the simplified formula is satisfiable ;
      - if this is the case, the original formula is satisfiable
      - otherwise, the same recursive check is done assuming the opposite truth value.
      - The simplification step uses unit propagation and pure literal elimination

SAT solving algorithms IV

3. CDCL (Conflict-Driven Clause Learning) **(Marques-Silva and Sakallah 1996)**
   - Iteratively set variables until either you find a satisfying assignment (done !) or you reach a conflict
   - When a conflict is reached, LEARN a clause that "remembers" the reason for the conflict.
   - Return UNSAT if the new learned clause is an empty clause

Today CDCL Solvers' features I



Today CDCL Solvers' features II

Two of the most important features of CDCL SAT solvers are **unit propagation** and **clause learning**.

Unit propagation

It is where the solver spends most of the time (more than 80%) when solving a formula. It uses **watched literals** for more efficiency.

Clause learning

It prevent solvers from repeatedly making the same errors while searching by generating and saving **asserting clauses** that capture the causes of encountered conflicts.

Simplification I

- **Subsumption :** A clause subsumes another clause if the first is included in the second. It retains the equivalence.
- **Elimination of a variable :** Application of the resolution on all the clauses where this variable appears. Does not preserve logical equivalence but only satisfiability.
- **Elimination of blocked clauses :** A clause is said to be blocked if it contains a blocked literal. Do not preserve logical equivalence but only satisfiability.
- **Elimination of hidden tautological clauses :** It is based on the extension of clauses by adding literals, called hidden literals. It is the opposite operation of self-subsumption. It preserves logical equivalence.
- **Elimination of equivalences :** A simple way to do this is to remove the equivalent literals.

Simplification II

- **Probing :** Analysis of the application of unit propagation on the two polarities of the same variable.
- **Vivification :** It aims to shorten the clauses of the formula. It consists of removing literals from the clause based on unit propagation.

Unit propagation

Consider the CNF formula
$\mathcal{F} = \{\neg a \vee b, \neg b \vee c, \neg d \vee \neg e, e \vee f, e \vee \neg f \vee \neg g, \neg h \vee \neg i, a, l \vee \neg e\}$ and the decision sequence $\delta = [d, h]$.

The propagation queue obtained after the assignment of the decision sequence literals is :

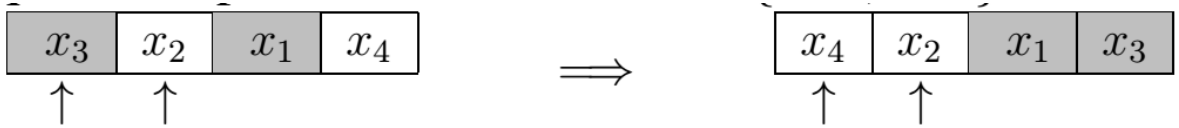
$$\mathcal{H} = \{\langle a, b, c\rangle, \langle (d), \neg e, f, \neg g\rangle, \langle (h), \neg i\rangle\}$$

Watched literals

Consider the clause $c = x_1 \vee x_2 \vee x_3 \vee x_4$. The watched literals are updated as follows :

1. Consider first the interpretation $\mathcal{I} = \{\neg x_1\}$

2. Suppose the interpretation is now $\mathcal{I} = \{\neg x_1, \neg x_3\}$



3. finally suppose the interpretation is $\mathcal{I} = \{\neg x_1, \neg x_3, \neg x_4\}$

The advantage of watched literals is that only a few number of clauses are visited when an assignment is made and no update is needed upon backtracking

Asserting clause generation I

Example

Consider the CNF formula $\mathcal{F}$ consisting of the following clauses :

$c_1 = \neg x_1 \vee \neg x_2$ $\quad c_2 = x_2 \vee x_3$ $\quad c_3 = \neg x_3 \vee x_4 \vee \neg x_5$

$c_5 = \neg x_6 \vee x_7 \vee x_8$ $\quad c_6 = x_5 \vee \neg x_8 \vee \neg x_9$ $\quad c_7 = x_9 \vee \neg x_{10}$

$c_9 = x_9 \vee \neg x_3 \vee x_{12}$ $\quad c_{10} = x_{10} \vee \neg x_{11} \vee \neg x_{13}$ $\quad c_{11} = \neg x_{11} \vee \neg x_{12} \vee x_{13}$

Consider further the interpretation $\mathcal{I}$ (decision literals are in bold) :

$$\mathcal{I} = \left(\mathbf{x_1^1}, \neg x_2^1, x_3^1, \neg \mathbf{x_4^2}, \neg x_5^2, x_6^2, \neg \mathbf{x_7^3}, x_8^3, \neg x_9^3, \neg x_{10}^3, x_{11}^3, x_{12}^3, x_{13}^3\right)$$

where the exponent represents the decision level of the corresponding literal.

Asserting clause generation II

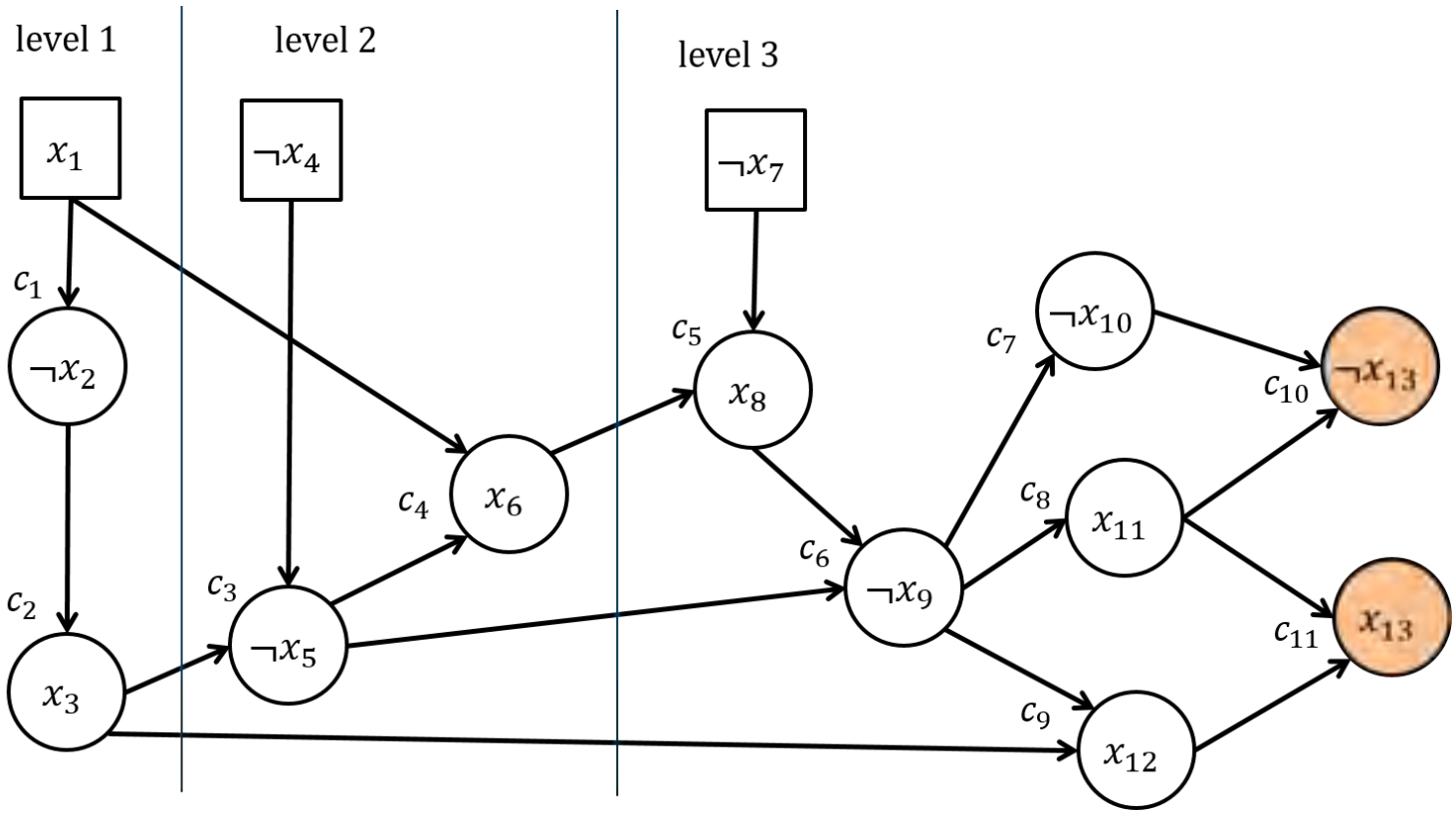


Figure – The implication graph associated with $\mathcal{F}$ and $\mathcal{I}$: Beside each node is indicated the reason clause of the associated literal. The graph shows a conflicting variable x_{13}

Asserting clause generation III

Example (Asserting clause generation : F-UIP)

The asserting clause R_4 is generated as follows :

$R_1 = c_{10} \bowtie_{x_{13}} c_{11} = x_{10}^3 \vee \neg x_{11}^3 \vee \neg x_{12}^3$

$R_2 = R_1 \bowtie_{x_{12}} c_9 = \neg x_3^1 \vee x_9^3 \vee x_{10}^3 \vee \neg x_{11}^3$

$R_3 = R_2 \bowtie_{x_{11}} c_8 = \neg x_3^1 \vee x_9^3 \vee x_{10}^3$

$R_4 = R_3 \bowtie_{x_{10}} c_7 = \neg x_3^1 \vee x_9^3$

- The asserting level of R_4 is 1 indicating that just after learning this clause, the solver should backtrack to level 1, set the asserting literal x_9 to true and continue the search ;
- After doing so and performing unit propagation, the interpretation becomes $\mathcal{I} = \left(\mathbf{x_1^1}, \neg x_2^1, x_3^1, x_9^1\right)$.

Heuristic of choice of variables

- **"Syntactic" heuristics :** The goal is to select the variables which, once assigned, generate as many propagations as possible or allow the most clauses to be satisfied.
- **Look-ahead heuristics :** They are based on the anticipation of the result of the assignment of a variable not yet assigned.
- **Look-back heuristics :** They take advantage of failures to return to relevant choice points (intelligent backtra-cking)
- **Dynamic heuristics driven by conflicts :** negligible computational cost and independence from current interpretation. Example : Variable State Independent Decaying Sum (VSIDS)

Polarity heuristics

- **false/true :** Assign false to a negative literal and true to a positive literal.
- **Jw :** It uses information on the problem (size and number of clauses containing the variable).
- **Progress saving :** The objective is to prevent the solver from solving a satisfiable sub-formula multiple times.
- **Occurence :** The main goal is to balance the number of positive and negative literals to allow more resolutions to be made

Parallel SAT Solving

Why parallelize the resolution of SAT ?

- Inability of sequential SAT solvers to meet the increasing performance demand ;
- Current limitation of microprocessor's frequency due to physical constraints
- Today's microprocessors power is increased by adding more cores.

Parallel Solving approaches

1. Divide-and-Conquer approaches ;
   - Formula partitioning : split the formula into several subsets, solve them and recombine sub-solutions ;
   - Search space partitioning : splits the search space, each solver being in charge of a particular subspace.
2. Portfolio approach : portfolio launches multiple solvers in parallel, and the first to find a solution ends the computation.

Formula Partitioning

1. The idea is to decompose the input problem into simpler sub-problems with less variables that can be independently solved.
2. Within a parallel framework, the list of variable solutions of each sub-problem are computed.
3. Finally, these partial interpretations are joined and checked to exhibit a global solution, if one exists.

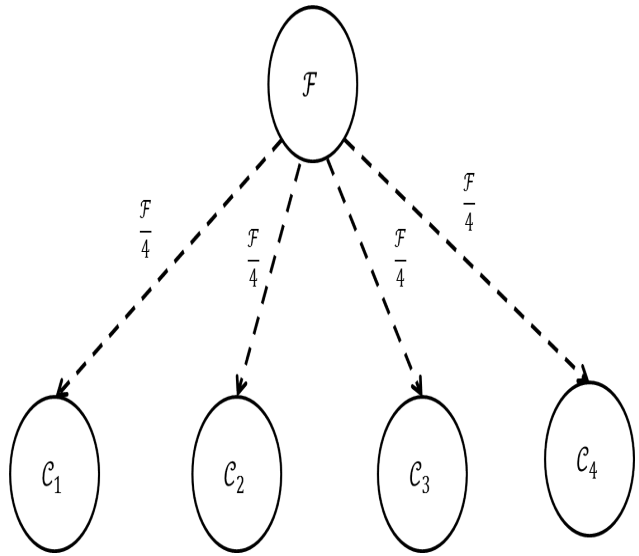


Figure – Formula partitioning

Search Space Splitting

- The search space is divided into multiple disjoint parts.
- Each part is assigned to a thread for solving,
- If one solver finds that the formula is satisfiable, then the search stops.
- A formula is declared unsatisfiable when all sub-formulas have been shown unsatisfiable.
- implementations use a master-slave architecture with work stealing for workload balancing.

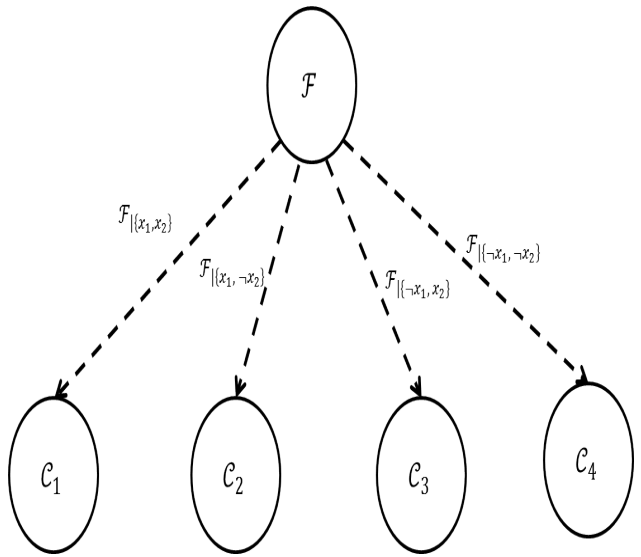


Figure – Search space splitting

Portfolio

- Run several incarnations of the same solver (with different configurations) on the input instance
- To be effective solvers should be tuned to use complementary configurations
- Information sharing : clauses, variable activities, equivalent variables ...
- The sharing helps improve the system over the best search strategy
- No load balancing needed

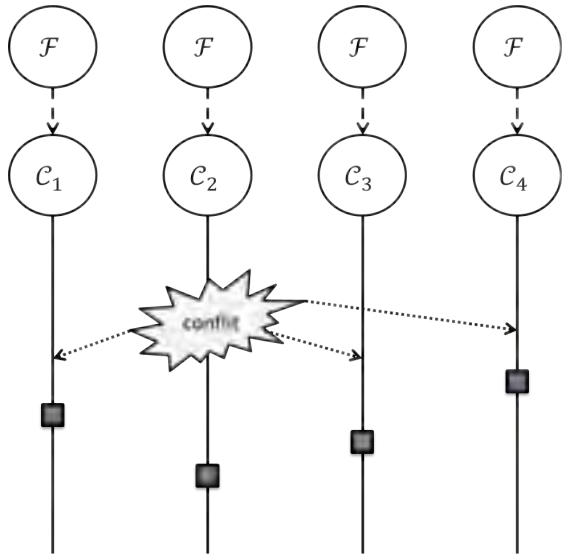


Figure – Portfolio approach

Techniques used in modern SAT solvers

- Preprocessing : The simplifications are applied before the problem solving starts**(Eén and Biere 2005)** ;
- Inprocessing : the simplification is carried out periodically during the actual search **(Järvisalo, Heule, and Biere 2012)**
- Symmetry breaking (static and dynamic) **(Crawford et al. 1996; Aloul et al. 2003; Benhamou et al. 2010; Bogaerts, Devriendt, and Bruynooghe 2017)**,
- Extended Resolution **(Tseitin 1968)**,
- Machine learning **(Liang et al. 2016; Haim and Walsh 2009)** ;

Some state-of-the-art SAT solvers

Sequential Solvers

- Minisat,
- Glucose,
- Maple_LCM_Dist_ChronoBT,
- Lingeling,
- CaDiCaL

Parallel Solvers

- ManySat,
- Glucose Syrup,
- PLingeling,
- PeneLoPe

For more information see the yearly SAT Competition Website (http ://www.satcompetition.org/)

Input Format

Before a problem can be solved using SAT technology, it must usually be encoded in CNF, allows a common file format for problems.

SAT solvers take as input a file containing the CNF formula to solve in DIMACS CNF format **(Challenge 1993)**.

```
c This is a comment line
p cnf 5 6
-1 -3 0
-2  3 0
 1  5 0
-1  2 4 0
 3 -4 0
 1 -5 0
```

Figure – DIMACS representation of the CNF formula
$\mathcal{F} = \{(\neg x_1 \vee \neg x_3), (\neg x_2 \vee x_3), (x_1 \vee x_5), (\neg x_1 \vee x_2 \vee x_4), (x_3 \vee \neg x_4), (x_1 \vee \neg x_5)\}$

Output of a SAT Solver

- The satisfiability status of the input formula (SATISFIABLE or UNSATISFIABLE)
- If SATISFIABLE, a model of the input formula is provided
- If UNSATISFIABLE, an UNSAT certificate is provided usually in DRUP or DRAT format. This certificate can be checked by an independent program.

Outline

SAT-based problem Solving I

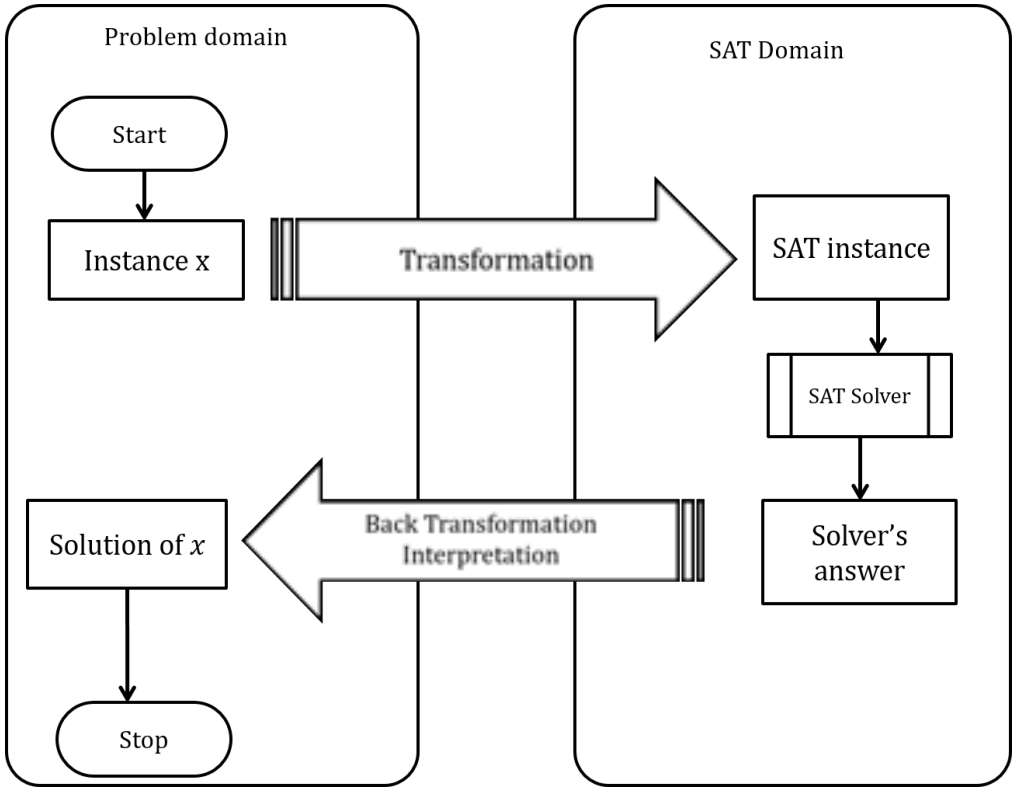


Figure – Solving a problem using a SAT solver

The transformation phase Uses cardinality constraints

CNF encoding of cardinality constraints I

- A cardinality constraint imposes that a certain number of variables must be true on a given set $X = \{x_1, x_2, x_3 \cdots, x_n\}$.
- Examples of carnality constraints :
  - AT most k : AtMost(k,X) $\equiv \sum_{i=1}^{n} x_i \leq k$;
  - At least k : AtLeast(k,X) $\equiv \sum_{i=1}^{n} x_i \geq k$;
  - Exactly k : Exactly(k ,X) $\equiv \sum_{i=1}^{n} x_i = k$.
- These constraints must be encoded in conjunctive normal form.

CNF encoding of cardinality constraints II

One constraint of great interest is the at most one (AMO) constraint.

Encoding Techniques

- Naive encoding
- Nested encoding
- 2-product encoding
- ...

Naive encoding AMO

The idea is to exclude each pair of variables satisfied simultaneously. We then obtain the following CNF formula :
$AMO(X) = \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j)$

CNF encoding of cardinality constraints III

Nested encoding **(Biere et al. 2014)**

The principle of this encoding is to reduce the number of clauses by recursively separating the constraint into two smaller constraints using auxiliary variables :

$\sum_{i=1}^{n} x_i \leq 1 \equiv (y + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} x_i \leq 1) \wedge (\neg y + \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} x_i \leq 1)$, where y is an auxiliary variable

CNF encoding of cardinality constraints IV

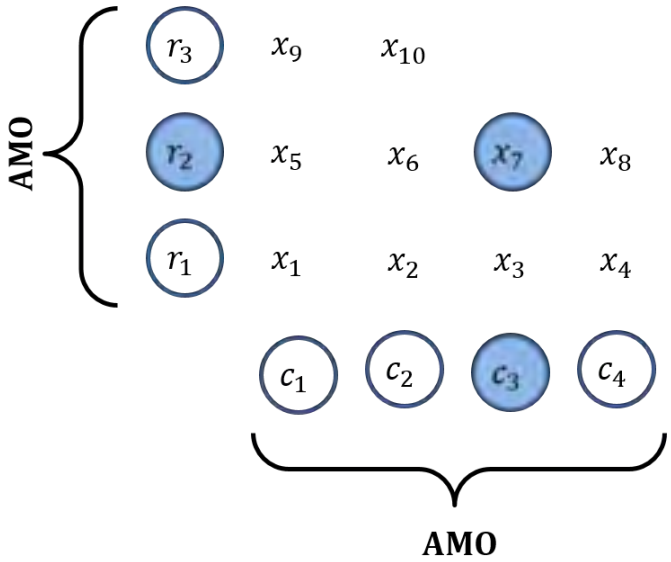
2-product encoding **(Chen 2010)**

This encoding makes use of a grid of n points corresponding to the n variables of the constraint, then uses a constraint to select at most one row and another constraint to select at most one column. Thus the point at their intersection selects the only variable of the constraint that must be « true ». Let $X = \{x_1, \ldots, x_n\}$, $p = \lceil \sqrt{n} \rceil$ and $q = \lceil n/p \rceil$:

$$AMO(X) = AMO(C) \wedge AMO(R) \bigwedge_{1 \leq i \leq p, 1 \leq j \leq q} \bigwedge_{1 \leq k \leq n, k = (i-1)q + j} (\neg x_k \vee c_i) \wedge (\neg x_k \wedge r_j)$$

where $C = \{c_1, \ldots, c_p\}$, $R = \{r_1, \ldots, r_q\}$; each element of C and R being an auxiliary variable.

CNF encoding of cardinality constraints V



Selection Clauses

$(\neg x_1 \vee r_1)$ $(\neg x_1 \vee c_1)$ $(\neg x_2 \vee r_1)$
$(\neg x_2 \vee c_2)$ $(\neg x_3 \vee r_1)$ $(\neg x_3 \vee c_3)$
$(\neg x_4 \vee r_1)$ $(\neg x_4 \vee c_4)$ $(\neg x_5 \vee r_2)$
$(\neg x_5 \vee c_1)$ $(\neg x_6 \vee r_2)$ $(\neg x_6 \vee c_2)$
$(\neg x_7 \vee r_2)$ $(\neg x_7 \vee c_3)$ $(\neg x_8 \vee r_2)$
$(\neg x_8 \vee c_4)$ $(\neg x_9 \vee r_3)$ $(\neg x_9 \vee c_1)$
$(\neg x_{10} \vee r_3)$ $(\neg x_{10} \vee c_2)$

Final *At Most One* Contraints

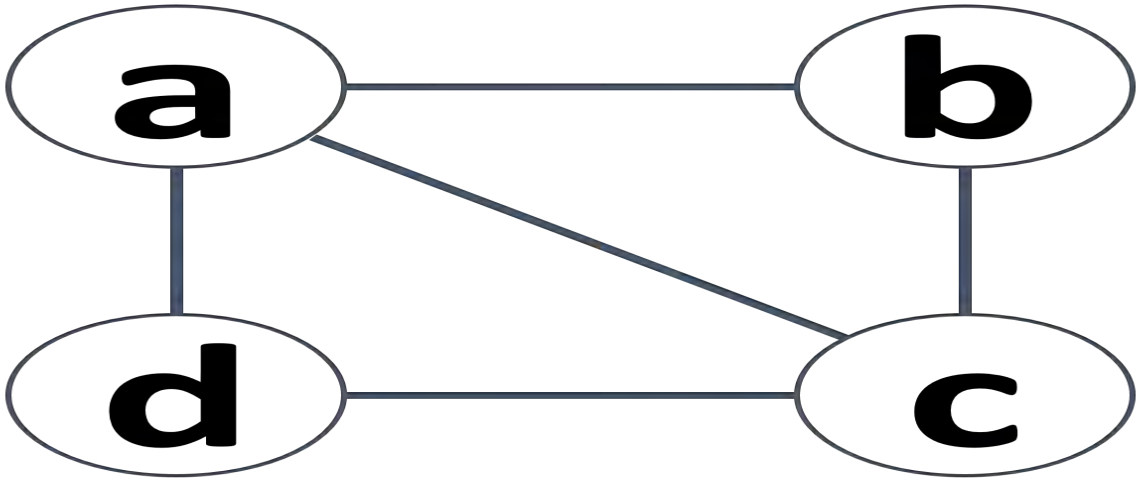
$(\neg r_1 \vee \neg r_2)$ $(\neg r_1 \vee \neg r_3)$ $(\neg r_2 \vee \neg r_3)$
$(\neg c_1 \vee \neg c_2)$ $(\neg c_1 \vee \neg c_3)$ $(\neg c_1 \vee \neg c_4)$
$(\neg c_2 \vee \neg c_3)$ $(\neg c_2 \vee \neg c_4)$ $(\neg c_3 \vee \neg c_4)$

Figure – Schematic representation of the *2-product encoding* for $\Sigma_{i=1}^{10} x_i \leq 1$

Example Graph coloring I

Graph coloring problem

Given a graph G and k different colors, is it possible to assign one of the k colors to each node of G such that no two adjacent nodes are assigned the same color ?



Example Graph coloring II

Figure – Can this graph be colored with 3 colors red, white and green ?

Choosing variables

Let Red = 1, white = 2 and Green = 3. Consider the following Boolean variables :

- $a_i, i \in \{1, 2, 3\}$ is true iff node a is colored with color i.
- $b_i, i \in \{1, 2, 3\}$ is true iff node b is colored with color i.
- $c_i, i \in \{1, 2, 3\}$ is true iff node c is colored with color i.
- $d_i, i \in \{1, 2, 3\}$ is true iff node d is colored with color i.

Each node must have at least one color

$a_1 \vee a_2 \vee a_3$; $b_1 \vee b_2 \vee b_3$; $c_1 \vee c_2 \vee c_3$; $d_1 \vee d_2 \vee d_3$

Example Graph coloring III

Each node must have at most one color

- $AMO(\{a_1, a_2, a_3\}) \equiv (\neg a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge (\neg a_2 \vee \neg a_3)$
- $AMO(\{b_1, b_2, b_3\}) \equiv (\neg b_1 \vee \neg b_2) \wedge (\neg b_1 \vee \neg b_3) \wedge (\neg b_2 \vee \neg b_3)$
- $AMO(\{c_1, c_2, c_3\}) \equiv (\neg c_1 \vee \neg c_2) \wedge (\neg c_1 \vee \neg c_3) \wedge (\neg c_2 \vee \neg c_3)$
- $AMO(\{d_1, d_2, d_3\}) \equiv (\neg d_1 \vee \neg d_2) \wedge (\neg d_1 \vee \neg d_3) \wedge (\neg d_2 \vee \neg d_3)$

Two adjacent nodes must have different colors

- for $i \in \{1, 2, 3\}$,
  $(a_i \Rightarrow \neg b_i \wedge \neg c_i \wedge \neg d_i) \equiv (\neg a_i \vee \neg b_i) \wedge (\neg a_i \vee \neg c_i) \wedge (\neg a_i \vee \neg d_i)$
- for $i \in \{1, 2, 3\}$, $(b_i \Rightarrow \neg a_i \wedge \neg c_i \equiv (\neg b_i \vee \neg a_i) \wedge (\neg b_i \vee \neg c_i)$
- for $i \in \{1, 2, 3\}$,
  $(c_i \Rightarrow \neg a_i \wedge \neg b_i \wedge \neg d_i) \equiv (\neg c_i \vee \neg a_i) \wedge (\neg c_i \vee \neg b_i) \wedge (\neg c_i \vee \neg d_i)$
- for $i \in \{1, 2, 3\}$, $(d_i \Rightarrow \neg a_i \wedge \neg c_i \equiv (\neg d_i \vee \neg a_i) \wedge (\neg d_i \vee \neg c_i)$

Example Graph coloring IV

SAT solver's result and interpretation

A SAT solver called with the resulting CNF formula produces the following model : $a_3 = true, b_2 = true, c_1 = true, d_2 = true$ with all the other variables set to false. This is interpreted as node a is green, node b is white, node c is red and node d is white.

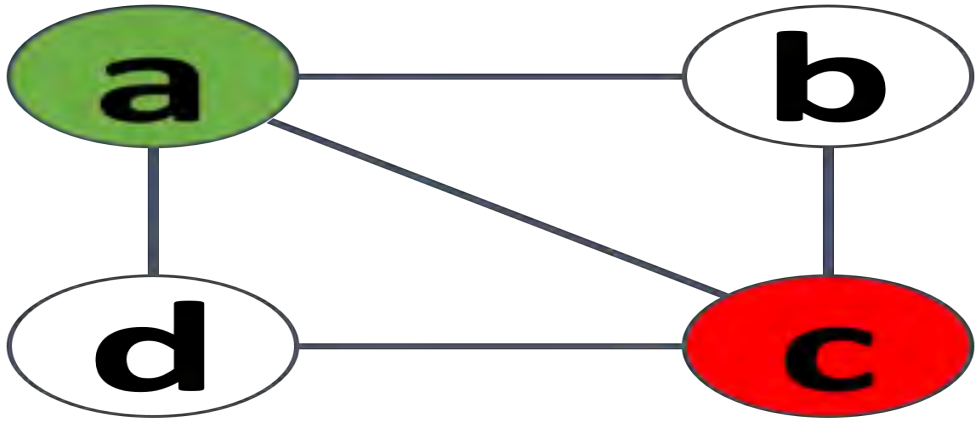


Figure – An example of coloring

Example Graph coloring V

Is it possible to color the same graph with 2 colors ?

The answer is NO, since the formula obtained by reducing it to SAT is unsatisfiable.

Success stories I

The Boolean Pythagorean Triples Problem (Cooper and Overstreet: 1980)

Is it possible to color each of the positive integers either red or blue, so that no Pythagorean triple of integers a, b, c, satisfying $a^2 + b^2 = c^2$ are all the same color ? For example, in the Pythagorean triple 3, 4 and 5 ($3^2 + 4^2 = 5^2$), if 3 and 4 are colored red, then 5 must be colored blue.

In the 1980s Ronald Graham offered a $100 prize for the solution of the problem.

The Largest Math Proof Ever

- 200 terabytes obtained by a SAT solver.
- About 2 days of calculation on a cluster with 800 cores.

Success stories II

Solution (Heule, Kullmann, and Marek 2016)

Up to 7,824, it is possible to color the integers, and even in several ways but, arrived at 7,825 and beyond, it becomes impossible.

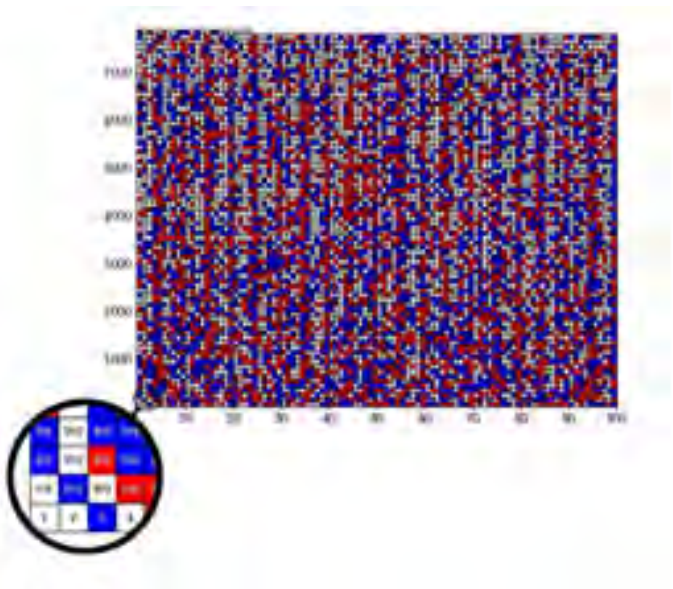


Figure – Grid showing one of the possible solutions of the Boolean Pythagorean Triples Problem for the numbers 1 to 7824

Other applications of SAT

- Cryptanalysis
- Software and hardware verification
- Software dependency management
- Timetable construction
- Graph coloring
- Games (N-queens, Sudoku)
- etc.

Summary

- SAT being an NP-complete problem which is currently solved quite well for many practical instance coming from industry,
- You don't need to be a SAT specialist to use SAT solvers,
- Parallelizing the resolution of SAT helps speed up its resolution,
- SAT is a mature technology that is used in industry.

Some interesting questions

- What is the optimal variable ordering for branching ?
- Which metric should be used to evaluate learned clauses quality an why should it be used ?
- How to integrate Extended resolution in CDCL Solvers to significantly improve practical SAT solving ?
- In parallel SAT solving, how to partition the search space so that the resulting subspace are of equal hardness ?
- In portfolio SAT solvers, how to perform the search so as to minimize redundancy ?
- Which new application domain can SAT technology be effective on ?

Some useful links

1. SAT competitions/Races/Challenges web site : `http://www.satcompetition.org/`,
2. Sparkle SAT Challenge : `http://ada.liacs.nl/events/sparkle-sat-18/`
3. Recent news about SAT : `http://www.satlive.org/`,
4. StarExec, a cross community logic solving service : `https://www.starexec.org/starexec/secure/index.jsp`

Thanks for your kind attention ! ! !

References I

- Aloul, Fadi A et al. (2003). "Solving difficult instances of Boolean satisfiability in the presence of symmetry". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22.9, pp. 1117–1137.
- Aspvall, Bengt, Michael F Plass, and Robert Endre Tarjan (1979). "A linear-time algorithm for testing the truth of certain quantified boolean formulas". In: *Information Processing Letters* 8.3, pp. 121–123.
- Benhamou, Belaıd et al. (2010). "Dynamic symmetry breaking in the satisfiability problem". In: *Proceedings of the 16th international conference on Logic for Programming, Artificial intelligence, and Reasoning. LPAR-16, Dakar, Senegal (April 25-may 1, 2010)*.
- Biere, Armin et al. (2014). "Detecting cardinality constraints in CNF". In: *Theory and Applications of Satisfiability Testing–SAT 2014*. Springer, pp. 285–301.

References II

- Bogaerts, Bart, Jo Devriendt, and Maurice Bruynooghe (2017). "Symmetric explanation learning : Effective dynamic symmetry handling for SAT". In: *Theory and Applications of Satisfiability Testing-SAT 2017, proceedings*. Vol. 10491. Springer, pp. 83–100.
- Challenge, DIMACS (1993). "Satisfiability : Suggested Format". In: *DIMACS Challenge. DIMACS*. url: `http://archive.dimacs.rutgers.edu/pub/challenge/sat/doc/satformat.tex`.
- Chen, Jingchao (2010). "A new SAT encoding of the at-most-one constraint". In: *Proc. Constraint Modelling and Reformulation*.
- Cook, Stephen A (1971). "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, pp. 151–158.
- Cooper, Joshua and Ralph Overstreet (2015). "Coloring so that no Pythagorean Triple is Monochromatic". In: *arXiv preprint arXiv :1505.02222*.

References III

- Crawford, James et al. (1996). "Symmetry-breaking predicates for search problems". In: *KR* 96, pp. 148–159.
- Davis, Martin, George Logemann, and Donald Loveland (1962). "A machine program for theorem-proving". In: *Communications of the ACM* 5.7, pp. 394–397.
- Davis, Martin and Hilary Putnam (1960). "A computing procedure for quantification theory". In: *Journal of the ACM (JACM)* 7.3, pp. 201–215.
- Deng, Tianyan and Daoyun Xu (2008). "NP-Completeness of (k-SAT, r-UNk-SAT) and (LSAT$\geq$ k, r-UNLSAT$\geq$ k)". In: *International Workshop on Frontiers in Algorithmics*. Springer, pp. 79–88.
- Ding-Zhu Du, Ker-I Ko (2000). *Theory of computational complexity*. Wiley. isbn: 0471345067,9780471345060.
- Dowling, William F and Jean H Gallier (1984). "Linear-time algorithms for testing the satisfiability of propositional Horn formulae". In: *The Journal of Logic Programming* 1.3, pp. 267–284.

References IV

- Eén, Niklas and Armin Biere (2005). "Effective preprocessing in SAT through variable and clause elimination". In: *Theory and Applications of Satisfiability Testing*. Springer, pp. 61–75.
- Haim, Shai and Toby Walsh (2009). "Restart Strategy Selection Using Machine Learning Techniques". In: *Theory and Applications of Satisfiability Testing - SAT 2009*. Ed. by Oliver Kullmann. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 312–325. isbn: 978-3-642-02777-2.
- Heule, Marijn JH, Oliver Kullmann, and Victor W Marek (2016). "Solving and verifying the boolean pythagorean triples problem via cube-and-conquer". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer, pp. 228–245.
- Järvisalo, Matti, Marijn JH Heule, and Armin Biere (2012). "Inprocessing rules". In: *International Joint Conference on Automated Reasoning*. Springer, pp. 355–370.

References V

- Krom, Melven R (1967). "The decision problem for a class of first-order formulas in which all disjunctions are binary". In: *Mathematical Logic Quarterly* 13.1-2, pp. 15–20.
- Liang, Jia Hui et al. (2016). "Learning rate based branching heuristic for SAT solvers". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer, pp. 123–140.
- Marques-Silva, JP and KA Sakallah (1996). *Grasp-a new search algorithm for satisfiability. ICCAD*.
- Tseitin, G (1968). "On the complexity ofderivation in propositional calculus". In: *Studies in Constrained Mathematics and Mathematical Logic*.