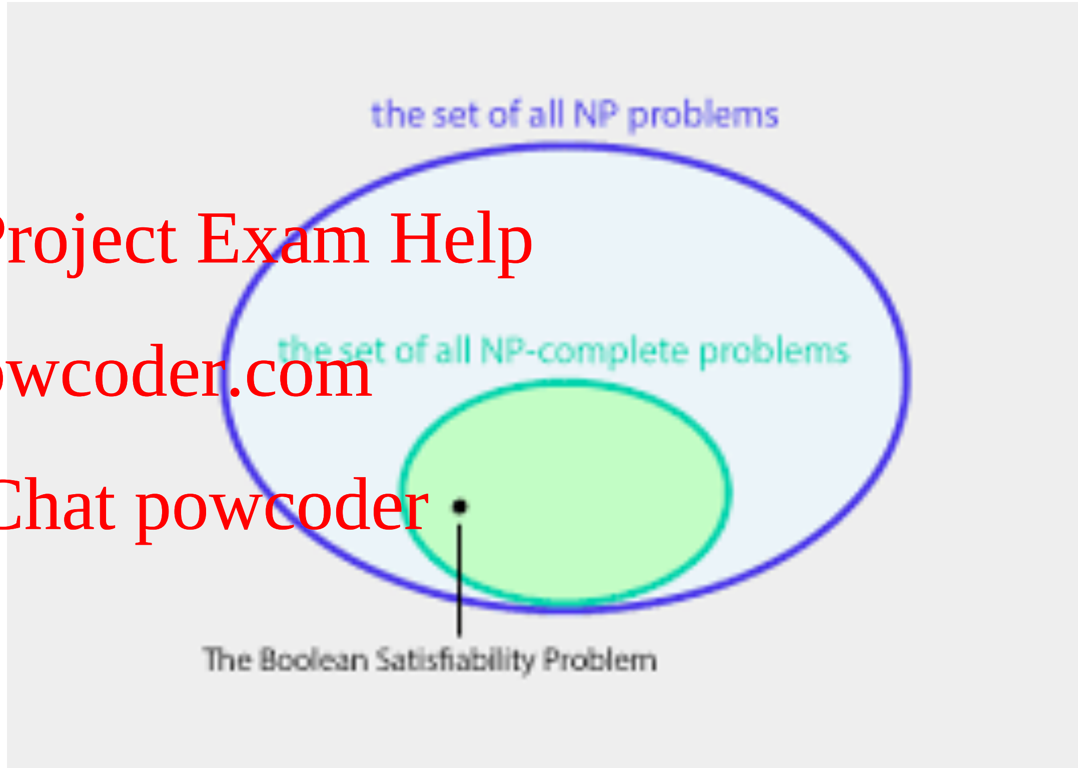
The SAT Problem

- Informal definition
- The SAT problem
- SAT algorithms
- Applications



Why is SAT important?

In theory

Canonical NP-Complete problem

In practice

Applied in many Computer Science problems:

- reachability analysis
- planning

- analysis of gene regulatory network

- fault diagnosis

SAT 2016 competition									
Organizers	Marijn Heule, Matti Järvisalo Tomáš Balyo								
Proceedings	Descriptions of the solvers and benchmarks								
Benchmarks	Available here								
Solvers	Available here								
	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
	Agile Track			Main Track			Random Track		
SAT+UNSAT	Riss	TB_Glucose	CHBR_Glucose	MapleCOMSPS	Riss	Lingeling	Dimetheus	CSCCSat	DCCAlm
	Parallel Track			No-Limit Track			Incremental Library Track		
SAT+UNSAT	Treengeling	Plingeling	CryptoMiniSat	BreakIDCOMiniSatPS	Lingeling	abcdSAT	CryptoMiniSat	Glucose	Riss
	Best Application Benchmark Solver in the Main Track			Best Crafted Benchmark Solver in the Main Track			Best Glucose Hack in the Main Track		
SAT+UNSAT	MapleCOMSPS			TC Glucose			Kiel		

What is the Boolean Satisfiability problem?

Informally

Given a propositional formula φ, the Boolean satisfiability (SAT) problem posed on φ means to determine whether there exists a *variable assignment* under which φ *evaluates to true*.

Formally

Let X be a set of propositional variables. Let F be a set of clauses over X.

- F is Satisfiable iff there exists a v: X →{0,1} s.t. $v \models F$
- F is Unsatisfiable iff $v \not\models F$, for all v: X →{0,1}

Example

$C1 = \{\neg x_1 \vee x_2, \neg x_2 \vee x_3\} = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$ Sat(C1) = Yes

$x_1 = 0, x_3 = 1,$

$C2 = C1 \cup \{x_1, \neg x_3\} = (x_3 \wedge \neg x_3)$ Sat(C2) = No

Why in CNF?

- There are efficient algorithms that can transform formulae into CNF.

- Most SAT solvers can exploit CNF
  - Easy to detect a conflict
  - Easy to remember partial assignments that don’t work (just add ‘conflict’ clauses)

It's a hard problem..!

A k-SAT problem with input in CNF with at most k literals in each clause…

- Complexity for different values of k:

  - 2-SAT: in P
  - 3-SAT: in NP-complete
  - > 3-SAT: is also NP-complete

A classification of SAT algorithms

- Complete algorithms
  - Proof systems - natural deduction, tableau, etc..
  - Davis-Putman (DP)
    - based on resolution
  - Stalmarck's method
  - Davis-Logemann-Loveland (DLL/DPLL)
    - search-based, basis for most successful solvers
  - Conflict-Driven Clause Learning (CDCL)

- Incomplete algorithms
  - Local search /hill climbing
  - Genetic algorithms

Notation and Terminology

- **Polarity of a variable** in a clause

  The "sign" with which the variable appears in the clause:

  $x_1 \vee \neg x_2$ x_1 positive polarity, x_2 negative polarity

- **Clause shorthand notations**

  Let $C = x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4$, we can write it

  $x_1x_2\neg x_3\neg x_4$ or as $x_1 + x_2 + \neg x_3 + \neg x_4$ or as $\{x_1,x_2,\neg x_3,\neg x_4\}$

- **Resolution**:

  Two clauses C1 and C2 that contain a variable x with opposite polarities, implies a new clause C3 that contains all literals except x and ¬x.

  $\{x_1,\neg x_2\}$ $\{x_2,\neg x_3\}$ $\longrightarrow$ $\{x_1,\neg x_3\}$

Davis-Putman (DP) Algorithm

Give a set S of clauses

For every variable in S

1. for every clause C_i containing the variable and every clause C_j containing the negation of the variable resolve C_i and C_j
2. add the resolvent to the set S of clauses
3. remove from S all original clauses containing the variable or its negation.

Two possible termination cases:

- Derive the empty clause (conclude UNSAT)
- All variables have been selected

Davis-Putman (DP): Example

Let S = {{p, q} {p, ¬q} {¬p, q} {¬p, ¬q} } choose p

$S' = \{\cancel{\{p, q\}}\ \cancel{\{p, \neg q\}}\ \cancel{\{\neg p, q\}}\ \cancel{\{\neg p, \neg q\}}\ \{q\}\ \{q, \neg q\}\ \{\neg q\}\ \}$

S' = { {q} {q, ¬q} {¬q} } choose q

What about tautologies?

$S'' = \{\ \cancel{\{q\}}\ \cancel{\{q, \neg q\}}\ \cancel{\{\neg q\}}\ \{q\}\ \{\neg q\}\ \{\}\}$

S'' = { {}} EMPTY CLAUSE: UNSAT

Davis-Putman (DP): Example 1

Let S = {{p, q} {p, ¬q} {¬p, q} {¬p, ¬q} } choose p

S' = { $\cancel{\{p}$, q} $\cancel{\{p}$, ¬q} $\cancel{\{\neg p}$, q} $\cancel{\{\neg p}$, ¬q} {q} {q, ¬q} {¬q} }

S' = { {q} {q, ¬q} {¬q} }

What about tautologies?

S'' = { {q} {q, ¬q} {¬q} {q} {¬q} {}}

S'' = { {}}

Eliminate tautologies first

S'' = { $\cancel{\{q\}}$ $\cancel{\{\neg q\}}$ } choose q

S'' = { {}} EMPTY CLAUSE: UNSAT

Davis-Putman (DP): Example 11

Let S = { {p,¬q} {q}} choose q

$S' = \{\{p, \cancel{\neg q}\}\ \cancel{\{q\}}\ \{p\}\}$

$S' = \{\{p\}\}$ No further variable can be chosen

Satisfiable and any assignment MUST make p true

BUT

- p appears only positively in the given set S of clauses.
- we could remove the clause that includes p from the beginning and “remember” that p has to be true.

$S = \{\ \cancel{\{p, \neg q\}}\ \{q\}\}$ [p]

$S = \{\cancel{\{q\}}\}$ [p, q]

$S = \{\ \}$ [p, q]

Basic Rules

Pure Literal rule

- Given a CNF formula F, expressed as a set S of clauses, a literal is pure in F if it only occurs only as a positive or as a negative literal in S.

$$S = \{\{\neg x_1, x_2\}\{x_3, \neg x_2\}\{x_4, \neg x_5\}\{x_5, \neg x_4\}\}$$

$\neg x_1$ and x_3 are pure literals

- Clauses containing pure literals can be removed from the formula (i.e. assign pure literals to the values that satisfy the clauses)

$$S = \{\cancel{\{\neg x_1, x_2\}}\ \cancel{\{x_3, \neg x_2\}}\{x_4, \neg x_5\}\{x_5, \neg x_4\}\}$$

$$S' = \{\{x_4, \neg x_5\}\{x_5, \neg x_4\}\}$$

Remember $[\neg x_1, x_3]$

Assignment must make x_1 = false and x_3 = true

Basic Rules

Unit clause

Clause with a single literal. It is satisfied by assuming the literal to be true.

Unit propagation rule

Once a literal has been assigned, its assignment needs to be propagated to other clauses:

- Every clause (other than the unit clause itself), that contains the unit clause, is removed

- In every clause that contains the negation of the unit clause, this negated literal is deleted

$$S = \{ \{x_1, x_2\} \ \{\neg x_1\ x_3\} \ \{\neg x_3\ x_4\} \ \{x_1\}\}$$

removed $\quad \neg x_1$ deleted $\quad$ unchanged $\quad$ unchanged

$$S' = \{ \{x_3\} \ \{\neg x_3\ x_4\} \ \{x_1\}\}$$

X_1 become then pure literal

Using a partial assignment: a clause can be unit clause when all literal except one are assigned based on a given partial assignment.

Improved Davis-Putman (DP) Algorithm

DP(M, S): M is a partial model of clauses processed so far
S is set of clauses still to process

DP([], S) (At the beginning)

Iteratively apply the following steps:

- Select a variable x
- Apply resolution rule between every pair of clauses of the form $(x \vee \alpha)$ and $(\neg x \vee \beta)$ in S

- Remove all clauses containing either x or ¬x
- Apply the pure literal rule and unit propagation

Terminate when either the empty clause or empty formula (equivalently, a formula containing only pure literals) is derived

Davis-Putman (DP) Algorithm II: example

$M = \{\}$

$S = \{\{x_1, \neg x_2, \neg x_3\}\{\neg x_1, \neg x_2, \neg x_3\}\{x_2, x_3\}\{x_3, x_4\}\{x_3, \neg x_4\}\}$

$S_1 = \{\{\neg x_2, \neg x_3\}\{x_2, x_3\}\{x_3, x_4\}\{x_3, \neg x_4\}\}$

$S_2 = \{\{\neg x_3, x_3\}\{x_3, x_4\}\{x_3, \neg x_4\}\}$ tautology

$S_3 = \{\{x_3, x_4\}\{x_3, \neg x_4\}\}$ x_3 pure literal

$S_4 = \{\}$ $M = \{x_3\}$

Satisfiable provided that x_3 is true.

Improved Davis-Putman (DP) Algorithm

Key ideas:

- Resolution eliminates 1 variable at each step
- Use of pure literal rule and unit propagation

But still inefficient…..

DLL (or DPLL) Algorithm

Basic idea:

- Instead of eliminating variables, split on a given variable at each step
- Also apply the pure literal rule and unit propagation

Basic Rules and Notations for DLL

Let F be the initial set of clauses and let M = {} be initial assignment.
Let Sc be the current set of clauses:

UNSAT(Sc)	If Sc contains the {}, then F is unsatisfiable
SAT(Sc)	If Sc = {} then F is satisfiable
MULT(L, Sc)	If a literal occurs more than once in a clause in Sc, then all but one can be deleted
SUB(C, Sc)	A clause C in Sc can be deleted, if it is a superset of another clause in Sc
TAUT(C, Sc)	Delete from Sc any clause that contains a literal L and its negation ¬L
PURE(L, Sc)	If L is a pure literal in Sc, delete all clauses that contain L, and add L to M (i.e. M = M ∪ {L}).

Basic Rules and Notations for DLL

Let F be the initial set of clauses and let M = {} be initial assignment.
Let Sc be the current set of clauses:

UNIT(L, Sc)	If Sc contain unit clause {L}, then delete all clauses in Sc that include L, remove the element ¬L from the remaining clauses, and add L to M (i.e. $M = M \cup \{L\}$).
SPLIT(L,Sc)	If Sc contains two clauses of the form $\{\ldots.\{C_k, L\}\ldots\{C_m, \neg L\}\ldots.\}$, branch the computation. • One branch has $M = M \cup \{L\}$ and Sc is generated as in UNIT rule assuming L to be the fact. • Other branch has $M = M \cup \{\neg L\}$ and Sc is generated as in UNIT rule assuming ¬L to be the fact.

DLL Procedure (pseudo code)

DLL(M, S): boolean;
%M is a (partial) model so far and S are clauses still to process

1. If SAT(S) return true; %M is a (partial) model
2. If UNSAT(S) return false; % S has no models
3. If SUB(C, S) return DLL(M, S \ C);
4. If PURE(L, S) return DLL([L | M], S'), S' = S \ {C_i | L in C_i}; %Make L true
5. If UNIT(L, S) return DLL([L | M], S'), S' generated from S by removing clauses containing L, and removing ¬L from rest.
6. If UNIT(¬L, S) return DLL([¬L | M], S'), S' generated from S by removing clauses containing ¬L, and removing L from rest.
7. Otherwise, SPLIT(L, S) return DLL([L | M], S') ∨ DLL([¬L | M], S''), S' formed as in Step 5 , and S" formed as in Step 6.

DLL Example

$S = \{\{x_1, x_2\}\ \{x_4, \neg x_2, \neg x_3\}\ \{\neg x_1, x_3\}\ \{\neg x_4\}\ \}$ $\quad M = \{\}$

UNIT($\neg x_4$, S) $\quad M = \{\neg x_4\}$

$S_1 = \{\{x_1, x_2\}\ \{\neg x_2, \neg x_3\}\ \{\neg x_1, x_3\}\}$

SPLIT(x_1, S_1) $\quad M = \{x_1, \neg x_4\}$

$S_1 = \{\{\neg x_2, \neg x_3\}\ \{x_3\}\}$

UNIT(x_3, S_1) $\quad M = \{x_3, x_1, \neg x_4\}$

$S_2 = \{\{\neg x_2\}\}$

PURE($\neg x_2$, S_2) $\quad M = \{\neg x_2, x_3, x_1, \neg x_4\}$

$S_3 = \{\ \}$

Satisfiable

SPLIT($\neg x_1$, S_1) $\quad M = \{\neg x_1, \neg x_4\}$

$S_1 = \{\{x_2\}\ \{\neg x_2, \neg x_3\}\}$

UNIT(x_2, S_1) $\quad M = \{x_2, \neg x_1, \neg x_4\}$

$S_2 = \{\{\neg x_3\}\}$

PURE($\neg x_3$, S_2) $\quad M = \{\neg x_3, x_2, \neg x_1, \neg x_4\}$

$S_3 = \{\ \}$

Satisfiable

Properties of DLL

- MULT, SUB, TAUT, UNIT rules are equivalence preserving

$$S' \equiv S$$

- PURE and SPLIT are unsatisfiability preserving.

  – PURE: S is unsatisfiable $\equiv$ S' is unsatisfiable

  – SPLIT: S is unsatisfiable $\equiv$ S' is unsatisfiable and S'' is unsatisfiable.

- Theorem
  - DLL(M, S) halts with False if S has no models.
  - DLL(M, S) halts with True and returns at least one (partial) model of S if S is satisfiable.

Summary

- Introduced the notion of satisfiability (SAT)
- Described simple DP algorithm, based on resolutions

- Introduced simple simplification rules

  - TAUT, SUB, PURE literals

- Unit propagation and SPLIT rules
- DLL/DPLL algorithm