DM841

Discrete Optimization

Satisfiability

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

SAT Problem

Satisfiability problem in propositional logic

$$
\begin{array}{l}
(x_5 \vee x_8 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_3 \vee \bar{x}_7) \wedge (\bar{x}_5 \vee x_3 \vee x_8) \wedge \\
(\bar{x}_6 \vee \bar{x}_1 \vee \bar{x}_5) \wedge (x_8 \vee \bar{x}_9 \vee x_3) \wedge (x_2 \vee x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_8 \vee x_4) \wedge \\
(\bar{x}_9 \vee \bar{x}_6 \vee x_8) \wedge (x_8 \vee x_3 \vee \bar{x}_9) \wedge (x_9 \vee \bar{x}_3 \vee x_8) \wedge (x_6 \vee \bar{x}_9 \vee x_5) \wedge \\
(x_2 \vee \bar{x}_3 \vee \bar{x}_8) \wedge (x_8 \vee \bar{x}_6 \vee \bar{x}_3) \wedge (x_8 \vee \bar{x}_3 \vee \bar{x}_1) \wedge (\bar{x}_8 \vee x_6 \vee \bar{x}_2) \wedge \\
(x_7 \vee x_9 \vee \bar{x}_2) \wedge (x_8 \vee \bar{x}_9 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_9 \vee x_4) \wedge (x_8 \vee x_1 \vee \bar{x}_2) \wedge \\
(x_3 \vee \bar{x}_4 \vee \bar{x}_6) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee x_5) \wedge (\bar{x}_7 \vee x_1 \vee x_6) \wedge (\bar{x}_5 \vee x_4 \vee \bar{x}_6) \wedge \\
(\bar{x}_4 \vee x_9 \vee \bar{x}_8) \wedge (x_2 \vee x_9 \vee x_1) \wedge (x_5 \vee \bar{x}_7 \vee x_1) \wedge (\bar{x}_7 \vee \bar{x}_9 \vee \bar{x}_6) \wedge \\
(x_2 \vee x_5 \vee x_4) \wedge (x_8 \vee \bar{x}_4 \vee x_5) \wedge (x_5 \vee x_9 \vee x_3) \wedge (\bar{x}_5 \vee \bar{x}_7 \vee x_9) \wedge \\
(x_2 \vee \bar{x}_8 \vee x_1) \wedge (\bar{x}_7 \vee x_1 \vee x_5) \wedge (x_1 \vee x_4 \vee x_3) \wedge (x_1 \vee \bar{x}_9 \vee \bar{x}_4) \wedge \\
(x_3 \vee x_5 \vee x_6) \wedge (\bar{x}_6 \vee x_3 \vee \bar{x}_9) \wedge (\bar{x}_7 \vee x_5 \vee x_9) \wedge (x_7 \vee \bar{x}_5 \vee \bar{x}_2) \wedge \\
(x_4 \vee x_7 \vee x_3) \wedge (x_4 \vee \bar{x}_9 \vee \bar{x}_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge \\
(x_6 \vee x_7 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_6 \vee x_2 \vee x_3) \wedge (\bar{x}_8 \vee x_2 \vee x_5)
\end{array}
$$

Does there exist a truth assignment satisfying all clauses?

Search for a satisfying assignment (or prove none exists)

SAT Problem

Satisfiability problem in propositional logic

$$
\begin{array}{l}
(x_5 \vee x_8 \vee \bar{x}_2) \wedge (x_2 \vee \bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_3 \vee \bar{x}_7) \wedge (\bar{x}_5 \vee x_3 \vee x_8) \wedge \\
(\bar{x}_6 \vee \bar{x}_1 \vee \bar{x}_5) \wedge (x_8 \vee \bar{x}_9 \vee x_3) \wedge (x_2 \vee x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_8 \vee x_4) \wedge \\
(\bar{x}_9 \vee \bar{x}_6 \vee x_8) \wedge (x_8 \vee x_3 \vee \bar{x}_9) \wedge (x_9 \vee \bar{x}_3 \vee x_8) \wedge (x_6 \vee \bar{x}_9 \vee x_5) \wedge \\
(x_2 \vee \bar{x}_3 \vee \bar{x}_8) \wedge (x_8 \vee \bar{x}_6 \vee \bar{x}_3) \wedge (x_8 \vee \bar{x}_3 \vee \bar{x}_1) \wedge (\bar{x}_8 \vee x_6 \vee \bar{x}_2) \wedge \\
(x_7 \vee x_9 \vee \bar{x}_2) \wedge (x_8 \vee \bar{x}_9 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_9 \vee x_4) \wedge (x_8 \vee x_1 \vee \bar{x}_2) \wedge \\
(x_3 \vee \bar{x}_4 \vee \bar{x}_6) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee x_5) \wedge (\bar{x}_7 \vee x_1 \vee x_6) \wedge (\bar{x}_5 \vee x_4 \vee \bar{x}_6) \wedge \\
(\bar{x}_4 \vee x_9 \vee \bar{x}_8) \wedge (x_2 \vee x_9 \vee x_1) \wedge (x_5 \vee \bar{x}_7 \vee x_1) \wedge (\bar{x}_7 \vee \bar{x}_9 \vee \bar{x}_6) \wedge \\
(x_2 \vee x_5 \vee x_4) \wedge (x_8 \vee \bar{x}_4 \vee x_5) \wedge (x_5 \vee x_9 \vee x_3) \wedge (\bar{x}_5 \vee \bar{x}_7 \vee x_9) \wedge \\
(x_2 \vee \bar{x}_8 \vee x_1) \wedge (\bar{x}_7 \vee x_1 \vee x_5) \wedge (x_1 \vee x_4 \vee x_3) \wedge (x_1 \vee \bar{x}_9 \vee \bar{x}_4) \wedge \\
(x_3 \vee x_5 \vee x_6) \wedge (\bar{x}_6 \vee x_3 \vee \bar{x}_9) \wedge (\bar{x}_7 \vee x_5 \vee x_9) \wedge (x_7 \vee \bar{x}_5 \vee \bar{x}_2) \wedge \\
(x_4 \vee x_7 \vee x_3) \wedge (x_4 \vee \bar{x}_9 \vee \bar{x}_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge (x_5 \vee \bar{x}_1 \vee x_7) \wedge \\
(x_6 \vee x_7 \vee \bar{x}_3) \wedge (\bar{x}_8 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_6 \vee x_2 \vee x_3) \wedge (\bar{x}_8 \vee x_2 \vee x_5)
\end{array}
$$

Does there exist a truth assignment satisfying all clauses?

Search for a satisfying assignment (or prove none exists)

Motivation

- SAT used to solve many other problems!
- Applications:
  Hardware and Software Verification, Planning, Scheduling, Optimal Control, Protocol Design, Routing, Combinatorial problems, Equivalence Checking, etc.
- From 100 variables, 200 constraints (early 90s)
  to 1,000,000 vars. and 20,000,000 clauses in 20 years.

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas
There are other types of logic: first-order logic, temporal logic, etc.

The proposition symbols x_1, x_2, etc. are sentences

If x is a sentence, $\neg x$ is a sentence (negation)
If x_1 and x_2 are sentences, $x_1 \wedge x_2$ is a sentence (conjunction)
If x_1 and x_2 are sentences, $x_1 \vee x_2$ is a sentence (disjunction)
If x_1 and x_2 are sentences, $x_1 \rightarrow x_2$ is a sentence (implication)
If x_1 and x_2 are sentences, $x_1 \leftrightarrow x_2$ is a sentence (biconditional)

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.

x_1	x_2	x_3
true	*true*	*false*

(With these symbols, 8 possible models, can be enumerated automatically.)

Simple recursive process evaluates an arbitrary sentence, e.g.,

$\neg x_1 \wedge (x_2 \vee x_3) = \mathit{true} \wedge (\mathit{false} \vee \mathit{true}) \Leftrightarrow \mathit{true} \wedge \mathit{true} \Leftrightarrow \mathit{true}$

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Logical equivalence

Two sentences are logically equivalent iff true in same models:
$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \rightarrow \beta) &\equiv (\neg\beta \rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \leftrightarrow \beta) &\equiv ((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)) && \text{bicond. elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validity and Satisfiability

A sentence is valid if it is true in all models,
e.g., $True$, $A \vee \neg A$, $A \rightarrow A$, $(A \wedge (A \rightarrow B)) \rightarrow B$

A sentence is satisfiable if it is true in some model
e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in no models
e.g., $A \wedge \neg A$

Conjunctive Normal Form

Every sentence in Propositional Logic is logically equivalent to a conjunction of clauses:

- A formula is in conjunctive normal form (CNF) iff it is of the form

$$\bigwedge_{i=1}^{m} \bigvee_{j=1}^{k_i} l_{ij} = (l_{11} \vee \ldots \vee l_{1k_1}) \wedge \ldots \wedge (l_{m1} \vee \ldots \vee l_{mk_m})$$

  where each literal l_{ij} is a propositional variable or its negation.

  The disjunctions of literlas: $c_i = (l_{i1} \vee \ldots \vee l_{ik_i})$ are called clauses.

- A formula is in k-CNF iff it is in CNF and all clauses contain exactly k literals (*i.e.*, for all i, $k_i = k$).
- In many cases, the restriction of SAT to CNF formulae is considered.
- For every propositional formula, there is an equivalent formula in 3-CNF.

Example:

$$
\begin{aligned}
F := \quad & \wedge (\neg x_2 \vee x_1) \\
& \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\
& \wedge (x_1 \vee x_2) \\
& \wedge (\neg x_4 \vee x_3) \\
& \wedge (\neg x_5 \vee x_3)
\end{aligned}
$$

- F is in CNF.
- Is F satisfiable?
  Yes, *e.g.*, $x_1 := x_2 := \top$, $x_3 := x_4 := x_5 := \bot$ is a model of F.

Conversion to CNF

$x_1 \leftrightarrow (x_2 \vee x_3)$

1. Eliminate $\leftrightarrow$, replacing $\alpha \leftrightarrow \beta$ with $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$.

   $(x_1 \rightarrow (x_2 \vee x_3)) \wedge ((x_2 \vee x_3) \rightarrow x_1)$

2. Eliminate $\rightarrow$, replacing $\alpha \rightarrow \beta$ with $\neg\alpha \vee \beta$.

   $(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg(x_2 \vee x_3) \vee x_1)$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

   $(\neg x_1 \vee x_2 \vee x_3) \wedge ((\neg x_2 \wedge \neg x_3) \vee x_1)$

4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:

   $(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_1)$

SAT Problem: The General Problem

SAT Problem (decision problem, search variant):

- **Given:** Formula F in propositional logic
- **Task:** Find an assignment of truth values to variables in F that renders F true, or decide that no such assignment exists.

SAT Problem: A Specific Problem Instance

- **Given:** Formula $F := (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
- **Task:** Find an assignment of truth values to variables x_1, x_2 that renders F true, or decide that no such assignment exists.

Special Cases

Not all instances are hard:

- Definite clauses: exactly one literal in the clause is positive. Eg:

  $$\neg\alpha \vee \neg\beta \vee \gamma$$

- Horn clauses: at most one literal is positive.

Easy interpretation: $\alpha \wedge \beta \rightarrow \gamma$ infers that $\neg\alpha \vee \neg\beta \vee \gamma$

Inference is easy by forward checking, linear time

Max SAT

Definition ((Maximum) K-Satisfiability (SAT))

Input: A set X of variables, a collection C of disjunctive clauses of at most k literals, where a literal is a variable or a negated variable in X.
k is a constant, $k > 2$.
Task: A truth assignment for X or a truth assignment that maximizes the number of clauses satisfied.

MAX-SAT (optimization problem)

Which is the maximal number of clauses satisfiable in a propositional logic formula F?

Outline

Notational Conventions

$$\Delta = \{\{X_1, \neg X_2, X_3\}, \{X_2, \neg X_4\}, \{\neg X_3, X_4, X_5\}\}$$

- Common boundary conditions that arise in recursive algorithms on CNFs:
  - A CNF Δ is valid if Δ is the empty set: $\Delta = \emptyset$.
  - A CNF Δ is inconsistent if Δ contains the empty set: $\emptyset \in A$.
- A formula Δ is said to imply another formula Γ, denoted $\Delta \models \Gamma$, iff every assignment that satisfies Δ also satisfies Γ.
- Note that if a clause C_i is a subset of another clause C_j, $C_i \subseteq C_j$, then $C_i \models C_j$, We say in this case that clause C_i subsumes clause C_j.
- if a CNF formula implies the empty clause, the formula is essentially unsatisfiable.

Resolution

One of the simplest complete algorithms for testing satisfiability is based on:

Definition (Resolution inference rule)

Let P be a Boolean variable, and suppose that Δ is a CNF which contains clauses C_i and C_j, where $P \in C_i$ and $\neg P \in C_j$. The resolution inference rule allows us to derive the clause $(C_i - \{P\}) \cup (C_j - \{\neg P\})$, which is called a resolvent that is obtained by resolving C_i and C_j.

1. $\{\neg P, R\}$
2. $\{\neg Q, R\}$
3. $\{\neg R\}$
4. $\{P, Q\}$

5.	$\{\neg P\}$	1, 3
6.	$\{\neg Q\}$	2, 3
7.	$\{Q\}$	4, 5
8.	$\{\}$	6, 7

- resolution is refutation complete on CNFs, i.e., it is guaranteed to derive the empty clause if the given CNF is unsatisfiable.
- complete algorithm for testing satisfiability: we keep applying resolution until either the empty clause is derived (unsatisfiable CNF) or until no more applications of resolution are possible (satisfiable CNF).

Unit Resolution

Definition (Unit Resolution)

Unit resolution is a special case of resolution, which requires that at least one of the resolved clauses has only one literal. Such clause is called a unit clause.

- Unit resolution is not refutation complete, which means that it may not derive the empty clause from an unsatisfiable CNF formula.
- Yet one can apply all possible unit resolution steps in time linear in the size of given CNF.

Conditioning

Conditioning Δ on $L = T$ yields a partition into three sets:

1. The set of clauses α containing the literal L. They become satisfied and can be removed from the formula.
2. The set of clauses α containing the literal $\neg L$. Since $\neg L$ is false we can remove the literal $\neg L$ from these clauses.
3. The set of clauses α that contain neither L nor $\neg L$.

That is:

$$\Delta \mid L = \{\alpha - \{\neg L\} \mid \alpha \in \Delta, L \notin \alpha\}$$

then

$$\Delta = \{\{A, B, \neg C\}, \{\neg A, D\}, \{B, C, D\}\},$$

and

$$\Delta|C = \{\{A, B\}, \{\neg A, D\}\},$$

$$\Delta|\neg C = \{\{\neg A, D\}, \{B, D\}\}.$$

Existential quantfication

The result of existentially quantifying variable P from a formula Δ is denoted by $\exists P \Delta$ and defined as follows:

$$\exists P \Delta := (\Delta \mid P) \vee (\Delta \mid \neg P)$$

- Δ is satisfiable iff $\exists P \Delta$ is satisfiable
- One can therefore existentially quantify all variables in the CNF Δ, one at a time, until we are left with a trivial CNF that contains no variables. It will be either $\{\emptyset\}$, unsatisfiable, or $\{\}$, valid, satifiable.

Solving SAT by Systematic Search: DPLL algorithm

Davis, Putam, Logenmann & Loveland (DPLL) algorithm is a recursive depth-first enumeration of possible models with the following elements:

1. Early termination (detect contradictions or success before reaching a leaf):
   a clause is true if any of its literals are true
   a formula is false if any of its clauses are false, which occurs when all its literals are false

2. Unit clause heuristic
   consider first unit clauses with just one literal or all literal but one already assigned. Generates cascade effect (forward chaining)

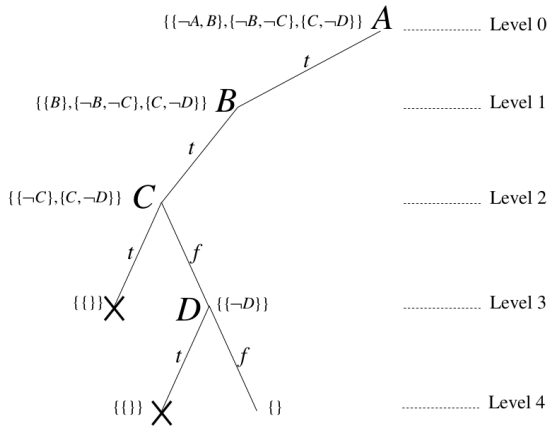
3. Pure literal heuristic:
   pure literal is one that appears with same sign everywhere.
   it can be assigned so that it makes the clauses true. Clauses already true can be ignored.

DPLL algorithm

Function DPLL(C, L, M):
> **Data:** C set of clauses; L set of literals; M model;
> **Result:** *true* or *false*
> **if** every clause in C is true in M **then return** *true*;
> **if** some clause in C is false in M **then return** *false*;
> $(\ell, val) \leftarrow$FindPureLiteral(L, C, M);
> **if** ℓ is non-null **then return** DPLL($C, L \setminus \ell, M \cup \{\ell = val\}$);
> $(\ell, val) \leftarrow$FindUnitClause(L, M);
> **if** ℓ is non-null **then return** DPLL($C, L \setminus \ell, M \cup \{\ell = val\}$);
> $\ell \leftarrow$First(L); $R \leftarrow$Rest(L);
> **return** DPLL($C, R, M \cup \{\ell = true\}$) or
> DPLL($C, R, M \cup \{\ell = false\}$)

- Modest memory requirements
- Branching can be seeing as conditioning: eg, at level d: $C \mid \ell_{d+1}$ and $C \mid \neg\ell_{d+1}$
- termination tree: the subset of search tree that is actually explored during search. In the worst case it is still $O(2^n)$

Termination Tree



Size and depth of tree appear to be good indicators of the problem character and difficulty.

Unit Resolution/Propagation

Unit resolution in DPLL:

- Before testing for success or failure, close the CNF under unit resolution and collect all unit clauses in the CNF.
- Then assume that variables are set to satisfy these unit clauses. That is, if the unit clause $\{P\}$ appears in the CNF, we set P to true. And if the unit clause $\{\neg P\}$ appears in the CNF, we set P to false.
- Simplify the CNF given these settings and check for either success (all clauses are subsumed) or failure (the empty clause is derived).

Hence:

Function `UnitResolution(`Δ`):`

Data: Δ a CNF

Result: l: a set of literals that were either present as unit clauses in Δ, or were derived from Δ by unit resolution; Γ: a new CNF which results from conditioning Δ on literals l

Example,

$$\Delta = \{\{\neg A, \neg B\}, \{B, C\}, \{\neg C, D\}, \{A\}\},$$

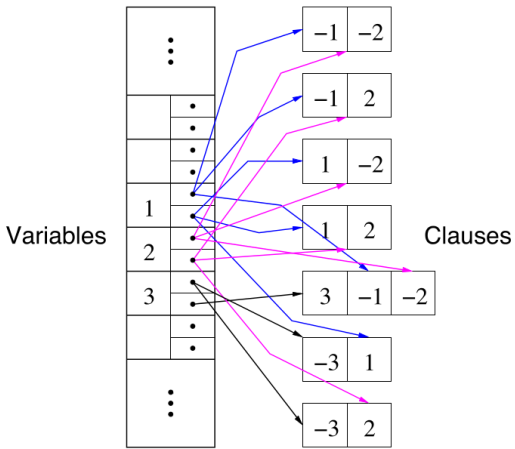
then $I = \{A, \neg B, C, D\}$ and $\Gamma = \{\}$.

$$\Delta = \{\{\neg A, \neg B\}, \{B, C\}, \{\neg C, D\}, \{C\}\}$$

then $I = \{C, D\}$ and $\Gamma = \{\{\neg A, \neg B\}\}$.

Speedups

- Component analysis to find separable problems
- Unit resolution/propagation
- Clever indexing (data structures)
- Variable-value ordering (See chp 8 of [BHMW])
- Random restarts
- Intelligent backtracking



Variable selection heuristics

Variable ordering or splitting heuristics:

- Degree
- Based on the occurrences in the (reduced) formula
  - Maximal Occurrence in clauses of Minimal Size (MOMS, Jeroslow-Wang)
- Variable State Independent Decaying Sum (VSIDS)
  - original idea (zChaff): for each conflict, increase the score of involved variables by 1, half all scores every 256 conflicts [MoskewiczMZZM2001]
  - improvement (MiniSAT): for each conflict, increase the score of involved variables by δ and increase $\delta := 1.05\delta$ [EenSörensson2003]

Value selection heuristics

Phase selection heuristics when choosing a particular literal of the selected variable:

- Based on the occurrences in the (reduced) formula
  - examples: Jeroslow-Wang, Maximal Occurrence in clauses of Minimal Size (MOMS), look-aheads

Available Solvers

DIMACS-CNF format: an input file in which each line represents a single disjunction. For example, a file with the two lines

```
1 -5 4 0
-1 5 3 4 0
```

represents the formula $(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4)$

Another common format for this formula is the 7-bit ASCII representation

```
(x1 | ~x5 | x4) & (~x1 | x5 | x3 | x4)
```

Solvers: MiniSAT, Lingeling, ... http://www.satcompetition.org/

Further Topics

- conflict-driven clause learning
- look-ahead

DPLL:

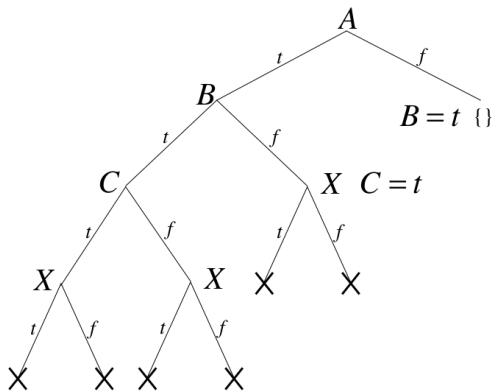
- performs chronological backtracking
- it does not take into account the information of the contradiction that triggers the backtrack.
- uses unit resolution, which is not refutation complete, hence it cannot necessarily detect the contradiction early on.

Enhancement:

- Non–chronological backtracking: backtracking to a lower level ℓ without necessarily trying every possibility between the current level and ℓ.
- conflict set: every assignment that contributes to the derivation of the empty clause
- backtrack to the most recent decision variable that appears in the conflict set and tries its different value. During this process, all intermediate assignments (between the current level and the backtrack level) are erased.

When backtracking goes past the conflict set it can still repeat the mistakes in the future. How to avoid this? $\rightsquigarrow$ clause learning

$\Delta =$
1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$



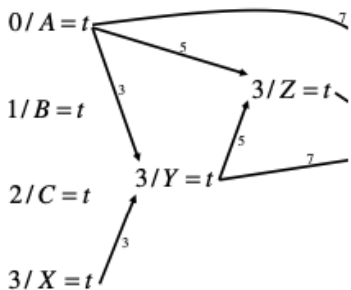
Contradictions that are discovered deep in the search tree are actually caused by having set A to true at Level 0

$A = \top, B = \top, C = \top, X = \top$, unit resolution will derive $Y = \top$ (Clause 3), $Z = \top$ (Clause 5), and detect that Clause 7 becomes empty (all literals are already $\bot$): conflict set is $\{A = \top, X = \top, Y = \top, Z = \top\}$.

Conflict Analysis

Implication graph: records dependencies among variable settings as they are established by unit resolution

$\Delta =$
1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$



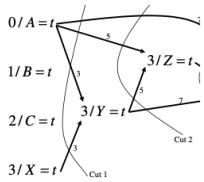
Nodes have the form $\ell/V = v$, which means that variable V has been set to value v at level ℓ either by decision or by an implication (decided by a unit resolution)

$\{A = \top, Y = \top, Z = \top\}$ used with Clause 7 derive the empty clause.

Conflict-driven clause

A Conflict set contains assignments that are sufficient to cause the conflict.

Every cut in the implication graph defines a conflict set as long as that cut separates the decision variables (root nodes) from the contradiction (a leaf node)



Conflict set contains any node (variable assignment) with an outgoing edge that crosses the cut.

Conflict–driven clause obtained by negating the assignments in the conflict set.

Example: conflict set $\{A = \top, Y = \top, Z = \top\}$, conflict–driven clause derived $\{\neg A, \neg Y, \neg Z\}$

Asserting conflict–driven clauses generated from cuts that contain exactly one variable assigned at the level where contradiction is found. Eg: $\{Y = \top, A = \top\}$ and $\{X = \top, A = \top\}$.

Learning a Conflict–Driven Clause and Backtracking

- Clause learning: From $\Delta \mid A, B, C, X$ unit resolution derives the asserting conflict-driven clause $\neg A \vee \neg X$
- the assertion level is the second highest level in a conflict–driven clause.
- undo all decisions made after the assertion level:
  in the clause $\neg A \vee \neg X$, A was set at Level 0 and X was set at Level 3. Hence, the assertion level is 0.
- Rationale: The assertion level is the deepest level at which adding the conflict–driven clause allows unit resolution to derive a new implication using that clause. (hence backtrack there, add the conflict-driven clause to the CNF, apply unit resolution, and continue the search.) far backtracking

Algorithm 3.5 DPLL+(CNF Δ): returns UNSATISFIABLE or SATISFIABLE.

```
1: D ← () {empty decision sequence}
2: Γ ← {} {empty set of learned clauses}
3: while true do
4:     if unit resolution detects a contradiction in (Δ, Γ, D) then
5:         if D = () then {contradiction without any decisions}
6:             return UNSATISFIABLE
7:         else {backtrack to assertion level}
8:             α ← asserting clause
9:             m ← assertion level of clause α
10:            D ← first m decisions in D {erase decisions ℓ_{m+1}, ...}
11:            add clause α to Γ
12:    else {unit resolution does not detect a contradiction}
13:        if ℓ is a literal where neither ℓ nor ¬ℓ are implied by unit resolution from (Δ, Γ, D)
           then
14:            D ← D; ℓ {add new decision to sequence D}
15:        else
16:            return SATISFIABLE
```

Further Elements

- Delete conflict-driven clauses:
  - new added conflict-driven clauses may subsume older conflict–driven clauses, which can be removed
  - if more deletion needed, then heuristic rules: eg, preference towards deleting longer, older and less active clauses.
- Restart by keeping conflict-based clauses

Certifying SAT Algorithms

- Verifying that a SAT solving algorithm produces the right result for satisfiable instances is straightforward
- In some applications, SAT algorithms are used to verify the correctness of hardware and software designs. Usually, unsatisfiability means that the design is free of certain types of bug.
- How to verify unsatisfiability results?
- Provide a resolution proof of the empty clause from the original set of clauses.
- All variables and clauses in the original formula are indexed. Each resolution is made explicit by listing the indices of the two operands, the index of the variable to be resolved on, (and the literals in the resolvent).
- alternatively, list all learned conflict-based clauses in order:
  if a learned clause C is derived from the CNF Δ, then applying unit resolution to $\Delta \wedge \neg C$ will result in a contradiction. The verifier may check to make sure that each conflict–driven clause can actually be derived from the set of preceding clauses by unit resolution.