Satisfiability

Michail G. Lagoudakis

Department of Computer Science
Duke University
Durham, NC 27708
Why SAT?

Historical Reasons

- The first NP-COMPLETE problem
- Important research challenge

Practical Reasons

- Many applications
- Many hard problems can be turned into SAT
- SAT solvers become better and better
Is SAT popular?

- SATLIB: [http://www.satlib.org](http://www.satlib.org)
- Google: “Satisfiability” returns 23,200 matches
- ResearchIndex: “Satisfiability” returns 2,027 documents
- SAT 2001: Workshop on Theory and Applications of SAT Testing
- SAT 2000: Third Workshop on the Satisfiability Problem
- Second DIMACS Implementation Challenge 1993
- Sat2000: Highlights of SAT Research in the Year 2000 (IOS, 2000)
- Algorithms for the Satisfiability Problem (CUP, 1999)
- Satisfiability Problem: Theory and Applications (AMS, 1997)
Overview

- The Satisfiability problem
- Algorithms for the Satisfiability problem
  - Complete Methods: DPLL, DP
  - Incomplete Methods: GSAT, WALKSAT
- Phase Transition in Random $k$-CNF
- SAT Technology
  - SATPLAN
  - BLACKBOX
Definitions

- **Boolean Variable**: \( x = \{ \text{FALSE, TRUE} \} = \{0, 1\} \)
- **Complement (Negation)**: \( \bar{x}, \neg x \)
  - \( \bar{x} = \text{TRUE}, \) if \( x = \text{FALSE} \)  \( \bar{x} = \text{FALSE}, \) if \( x = \text{TRUE} \)
  - \( \bar{x} = \neg(\neg x) = x \)
- **Literals**: \( x \) and \( \bar{x} \)
- **Disjunction (OR)**: \( (l_1 + l_2) \quad (l_1 \lor l_2) \quad (l_1 + l_2 + l_3 + \ldots + l_n) \)
- **Conjunction (AND)**: \( (l_1 l_2) \quad (l_1 \land l_2) \quad (l_1 l_2 l_3 \ldots l_n) \)
- **De Morgan Laws**: \( \overline{(l_1 + l_2)} = (\bar{l_1} \bar{l_2}) \quad \overline{(l_1 l_2)} = (\bar{l_1} + \bar{l_2}) \)
- **Boolean Formula**: \( xy + \bar{z}yw + (x + \bar{y})z \)
The Satisfiability (SAT) Problem

Given

- A boolean formula over \( n \) variables.

Find

- A satisfying assignment, that is, an assignment to all variables such that the formula evaluates to TRUE.

The formula can be unsatisfiable, satisfiable, or a tautology.

Example

- \( (x_1 + x_2 + \overline{x}_3)(\overline{x}_2 + x_4)(\overline{x}_1 + \overline{x}_5) \)

- \( (\overline{x}_1 + x_2 + x_3 + \overline{x}_4 + x_5)(\overline{x}_3)(x_2 + x_5)(\overline{x}_5)(x_1) \)

- \( (x_1, x_2, x_3, x_4, x_5) = (1, 1, 0, 1, 0) \)
Normal Forms

Conjunctive Normal Form (CNF)

- A conjunction of disjunctions (clausal form)
- Example: \((x_1 + x_2 + \bar{x}_3)(\bar{x}_2 + x_4)(\bar{x}_1 + \bar{x}_5)\)

\(k\)-CNF

- A CNF formula where each clause contains exactly \(k\) literals.
- Example (3-CNF): \((x_1 + x_2 + \bar{x}_3)(\bar{x}_2 + x_3 + x_4)(\bar{x}_1 + x_2 + \bar{x}_5)\)

Disjunctive Normal Form (DNF)

- A disjunction of conjunctions
- Example: \(x_1x_2\bar{x}_3 + \bar{x}_2x_4 + \bar{x}_1\bar{x}_5 + x_1\bar{x}_2x_3\bar{x}_4x_5\)
SAT and Variations

Given a CNF boolean formula over $n$ variables:

- **Decision SAT**: Is there a satisfying assignment?
- **SAT**: Find a satisfying assignment, if such an assignment exists.
- **#SAT**: Find the number of satisfying assignments.
- **MAJ-SAT**: Is the majority of the assignments satisfying?
- **MAX-SAT**: Find an assignment that maximizes the number of satisfied clauses.

These are all HARD problems!
SAT as CSP

SAT is a Constraint Satisfaction Problem!

- **Variables**: Boolean variables in the formula
- **Domains**: Each variable can take values from \( \{0, 1\} \)
- **Constraints**: Each clause of size \( k \) is a \( k \)-ary constraint

\[
(\overline{x}_2 + x_3 + x_4)
\]

\( x_2, x_3, \) and \( x_4 \) cannot be simultaneously \( x_2 = 1, x_3 = 0, \) and \( x_4 = 0 \)

- **Solution**: An assignment that satisfies all constraints (clauses).
Algorithms for the Satisfiability Problem

Complete or Systematic Methods

- Explore the space of all possible assignments systematically.
- If there is a satisfying assignment, it will be found.
- If no satisfying assignment is found, the formula is unsatisfiable.

Incomplete or Stochastic Methods

- Stochastic moves in the space of all possible assignments.
- If there is a satisfying assignment, it may be found.
- If no satisfying assignment is found, the formula may or may not be unsatisfiable.
The DPLL procedure

DPLL stands for Davis-Putnam-Logemann-Loveland (1962)

SAT as a Search Problem

- *State Space*: Space of all partial assignments.
- *Initial State*: All variables unassigned.
- *Actions*: Assign a variable to TRUE or FALSE.
- *Goal State*: A satisfying assignment (partial or complete).

Some Facts

- Depth-first search with branching factor 2.
- Partial assignments that cause contradiction can be pruned.
- Interested in the goal only, not in the path to the goal.
Unit Propagation

- If there is a unit clause, there is only one promising assignment to the corresponding variable (unary constraint).
- Example: \((x_1 + x_2 + x_3)(\overline{x}_2)(\overline{x}_1 + \overline{x}_5)(\overline{x}_2 + x_4)\)
- Make that assignment \((x_2 = 0)\) and eliminate the variable \((x_2)\).

Purification

- A pure variable appears purely in positive \((x_i)\) or negative \((\overline{x}_i)\) form.
- Assign \(x_i = 1\) in the positive case and \(x_i = 0\) in the negative case.
- Example: \(x_3\) and \(x_5\) are pure. Assign \(x_3 = 1\) and \(x_5 = 0\).
DPLL: Branching

Branching or Splitting

- If there are not unit clauses or pure variables, select an unassigned variable and try in turn the two possible assignments.
- Create a reduced formula in each case and continue recursively.

Example

- \((x_1 + x_2 + x_3)(\overline{x}_1 + \overline{x}_5)(\overline{x}_2 + x_4)(x_1 + \overline{x}_3 + x_5)\)
- Assume that \(x_1\) is selected.
- \(x_1 = 1\) gives \((\overline{x}_5)(\overline{x}_2 + x_4)\)
- \(x_1 = 0\) gives \((x_2 + x_3)(\overline{x}_2 + x_4)(\overline{x}_3 + x_5)\)
DPLL: Pseudocode

DPLL($F$)
    if ($F$ contains an empty clause)
        return “unsatisfiable”
    if ($F$ is empty)
        output current assignment
        return “satisfiable”
    /* Unit Propagation */
    if ($F$ contains a unit clause \{\textit{l}\})
        Create $F'$ from $F$ by eliminating all clauses that contain \textit{l} and all appearances of $\overline{\textit{l}}$
        return DPLL($F'$)
    /* Purification */
    if ($F$ contains a pure literal \textit{l})
        return DPLL($F$ \cup \{\textit{l}\})
    /* Branching */
    *** Select a free literal \textit{l} ***
    if (DPLL($F$ \cup \{\textit{l}\}) is “satisfiable”)
        return “satisfiable”
    else
        return DPLL($F$ \cup \{\overline{\textit{l}}\})
DPLL: Branching Heuristics

- Selecting a free literal is a non-deterministic step.
- Several branching heuristics have been developed:
  - Select the literal with the maximum number of occurrences.
  - Select the literal with maximum occurrences in minimum size clauses.
  - Select the literal that will cause the most unit propagations.
- The branching choices can have a tremendous impact on the size of the search tree.
- Example: 3CNF(100,430) : 101, 961, 963, 2493, 3638, 41248, 169048.
- Making “optimal” branching choices is NP-HARD (not surprising!).
- Remember the cutsets for CSPs?
Resolution

Resolution Inference Rule

- Given that \((x + y)(\bar{y} + z)\), infer \((x + z)\).

Variable Elimination

- Given a CNF formula \(F\)
- Find a variable \(x\) that appears in both positive and negative form.
- If there is no such variable, we are done! Why?
- Let \(F^+(x) \subset F\) be the set of clauses of \(F\) that contain \(x\).
- Let \(F^-(x) \subset F\) be the set of clauses of \(F\) that contain \(\bar{x}\).
- Create \(RES(x) = \{res_x(c_1, c_2) \mid c_1 \in F^+(x), c_2 \in F^-(x)\}\).
- \(F\) is equivalent to \((F - F^+(x) - F^-(x)) \cup RES(x)\)
**Variable Elimination Example**

- \( F = (x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_2 + x_4)(\overline{x}_1 + \overline{x}_3)(\overline{x}_2 + x_3 + x_4)(\overline{x}_1 + x_2 + \overline{x}_5) \)
- Select \( x_1 \)
- \( F^+ = (x_1 + \overline{x}_2 + \overline{x}_3) \)
- \( F^- = (\overline{x}_1 + \overline{x}_3)(\overline{x}_1 + x_2 + \overline{x}_5) \)
- \( F^{n+} = (\overline{x}_2 + \overline{x}_3) \)
- \( F^{n-} = (\overline{x}_3)(x_2 + \overline{x}_5) \)
- \( F = (\overline{x}_2 + x_4)(\overline{x}_2 + x_3 + x_4)(\overline{x}_2 + \overline{x}_3 + \overline{x}_3)(\overline{x}_2 + \overline{x}_3 + x_2 + \overline{x}_5) \)
- \( F = (\overline{x}_2 + x_4)(\overline{x}_2 + x_3 + x_4)(\overline{x}_2 + \overline{x}_3) \)
- Select ...
The Davis-Putnam Procedure (1960)

\[
\text{DP}(F)
\]

if \( F \) contains an empty clause
    \text{return} \ “unsatisfiable”
if \( F \) is empty
    \text{return} \ “satisfiable”

/* Unit Propagation */
if \( F \) contains a unit clause \( \{l\} \)
    Create \( F' \) from \( F \) by eliminating all clauses that
    contain \( l \) and all appearances of \( \bar{l} \)
    \text{return} \ \text{DP}(F')

/* Purification */
if \( F \) contains a pure literal \( l \)
    \text{return} \ \text{DP}(F \cup \{l\})

/* Resolution */
*** Select a free variable \( x \) ***
\[
F' = (F - F^+ (x) - F^- (x)) \cup RES(x)
\]
\text{return} \ \text{DP}(F')
Variable Ordering in DP

- Selecting a free variable to resolve is a non-deterministic step.

- Eliminating one variable can possibly create quadratically many clauses.

- The ordering of variables can have a tremendous impact on the size of the resulting formulas.

- Making “optimal” ordering choices is NP-HARD (not surprising!).

- Remember the elimination ordering in CSPs?
Satisfiability

**Complexity Issues**

**DPLL**

- Linear Space (current partial assignment, active clauses)
- In general, exponential time (binary search tree).
- Worst case is exponential.

**DP**

- Linear elimination steps (each step eliminates one variable).
- In general, exponential space is needed to store the formula.
- Worst case is exponential.

Preference is given to DPLL for it is easier to implement.
SAT as Local Search

- **State Space**: Space of all full assignments.
- **Initial State**: A random assignment.
- **Actions**: Flip one variable in the assignment.
- **Goal State**: A satisfying assignment (partial or complete).
- **Score**: The number of satisfied clauses
- **Objective**: Maximize score.
GSAT

Selman, Levesque, and Mitchell (1992)

GSAT($F'$)

for ($i=1$ to MAXTRIES)

Select a complete random assignment $A$

for ($j=1$ to MAXFLIPS)

if ($A$ satisfies all clauses in $F'$)

return “satisfiable”, $A$

else

Flip a variable that maximizes the score

(score = number of satisfied clauses)

Flip at random if no variable flip increases the score

What does it do?

• Hill Climbing with Random Restarts and Sideway Moves
Satisfiability

WALKSAT

Selman, Kautz, and Kohen (1994)

WALKSAT(F)

\[ \text{for } (i=1 \text{ to } \text{MAXTRIES}) \]
\[ \text{Select a complete random assignment } A \]
\[ \text{for } (j=1 \text{ to } \text{MAXFLIPS}) \]
\[ \text{if } (A \text{ satisfies all clauses in } F) \]
\[ \text{return } \text{“satisfiable”}, A \]
\[ \text{else} \]
\[ \text{With probability } p \text{ /* GSAT */} \]
\[ \text{Flip a variable that maximizes the score} \]
\[ \text{(score = number of satisfied clauses)} \]
\[ \text{Flip at random if no variable flip increases the score} \]
\[ \text{With probability } 1 - p \text{ /* Random Walk */} \]
\[ \text{Pick an unsatisfied clause } C \]
\[ \text{Flip a randomly chosen variable in } C \]
GSAT, WALKSAT, and more

- Surprisingly efficient!
- Major breakthrough in SAT research
- WALKSAT rendered the DIMACS benchmark library obsolete!
- Promising alternative to systematic methods.
- Cannot show unsatisfiability.
- Other incomplete methods:
  - Neural Networks
  - Genetic Algorithms
  - Simulated Annealing
Combinatorics

- Fix $n$ (variables), $m$ (clauses), $k$ (clause length).
- Generate $m$ clauses by choosing $k$ variables at random.
- In each clause, negate the resulting variables randomly (flip an unbiased coin).
- Is it possible to express the number of possible formulas as a function of $n$, $m$, $k$?
Randomly Generated $k$-CNF

**Combinatorics**

- Fix $n$ (variables), $m$ (clauses), $k$ (clause length).
- Generate $m$ clauses by choosing $k$ variables at random.
- In each clause, negate the resulting variables randomly (flip an unbiased coin).
- Is it possible to express the number of possible formulas as a function of $n$, $m$, $k$?

$$\left(\frac{n}{k} \times 2^k\right)^m$$

It may generate some duplicate clauses.
Properties

- $F$ is a randomly generated $k$-CNF.
- What is the probability that $F$ is satisfiable/unsatisfiable?
- It depends on the ratio $m/n$:
  - $m/n < 4.2$, $F$ is underconstrained; most likely satisfiable.
  - $m/n > 4.2$, $F$ is overconstrained; most likely unsatisfiable.
  - $m/n \approx 4.2$, $F$ is critically constrained; need to search.
- Easy-Hard-Easy distribution of instances.
- Phase transition phenomena at $m/n \approx 4.26$.
- The transition sharpens as $n$ increases.
Random $\kappa$-CNF Distribution

percent satisfiable

variables

- 50
- 40
- 30
- 20
- 10

ratio of clauses to variables

0 1 2 3 4 5 6 7 8
SATPLAN

Henry Kautz and Bart Selman (1996)

Idea

- Transform a planning problem into a satisfiability problem.
- Use a general-purpose SAT solver to find a satisfying assignment.
- Translate the satisfying assignment back to a plan for the original problem.

Results

- Efficient.
- Key issue: SAT encoding of the planning problem.
- Huge SAT instances (around 10,000 variables)
Henry Kautz and Bart Selman (1999)

**Idea**

- BLACKBOX is a combination of SATPLAN and GRAPHPLAN.
- Transform a planning problem into a plan graph of length $k$.
- Apply GRAPHPLAN simplifications.
- Convert the plan graph to a CNF formula.
- Apply SATPLAN simplifications.
- Solve the resulting CNF using any SAT solver.
- If a satisfying assignment is found, convert to a plan.
- Otherwise, increase $k$ and repeat.
In Conclusion

- SAT and its variations are HARD problems.
- There is a lot of active research in satisfiability.
- There are many practical applications of SAT.
- SAT solvers improve consistently.
- SAT is still a BIG research challenge.