An Introduction to SAT Solving

Tomáš Balyo

December 2, 2010

Outline

1 Definition & Motivation
- What is SAT?
- Why do we need to solve SAT?

2 Solving SAT
- DPLL
- Parallelization

3 Graphs and SAT
- Graph Representations of SAT Instances
- Connected Components of SAT Formulas

Outline

1. Definition & Motivation
   - What is SAT?
   - Why do we need to solve SAT?
2. Solving SAT
   - DPLL
   - Parallelization
3. Graphs and SAT
   - Graph Representations of SAT Instances
   - Connected Components of SAT Formulas

Some Definitions

- A Boolean variable x has 2 possible values: *True* and *False*.
- A literal is a Boolean variable (x) or its negation ($\neg x$).
- A clause is a disjunction of literals ($(x_1 \vee x_5 \vee \neg x_6)$).
- A CNF formula is a conjunction of clauses ($(x_1 \vee x_5 \vee \neg x_6) \wedge (x_2 \vee \neg x_5 \vee x_2) \wedge (x_4 \vee \neg x_6)$).

Some Definitions 2

- A truth assignment assigns a value (*True* or *False*) to each Boolean variable.
- A positive literal is *Satisfied* iff its variable has the value *True*. A negative literal is *Satisfied* iff its variable has the value *False*. A clause is *Satisfied* iff at least one of its literals is *Satisfied*. A CNF formula is *Satisfied* iff all ot its clauses are *Satisfied*.
- A CNF formula is called satisfiable if there is such a truth assignment to its variables, that the formula is *Satisfied*.
- The Boolean satisfiability problem (SAT) is the problem of determining whether a given CNF formula is satisfiable or not.

Some Properties

- SAT in NP-complete
- No deterministic algorithm solving SAT faster than $2^{\# variables}$ (in the worst case) has been found yet.
- But many SAT instances (with millions of variables) can be solved very quickly using the current state-of-the-art SAT solvers.

Outline

1 Definition & Motivation
- What is SAT?
- Why do we need to solve SAT?

2 Solving SAT
- DPLL
- Parallelization

3 Graphs and SAT
- Graph Representations of SAT Instances
- Connected Components of SAT Formulas

Applications

- Hardware and software verification
  - is it possible? (halting problem) - bounded model checking (BMC)
  - is not testing simpler and more efficient? - The FDIV Bug (1994)

$\frac{4195835}{3145727} = 1.333820$ (Correct) $\frac{4195835}{3145727} = 1.333739$ (FDIV result)

- A.I. problems are translated to SAT
  - planning (SATPLAN)
  - automated reasoning
- Haplotyping in Bioinformatics (identification of haplotypes)

Outline

1 Definition & Motivation
- What is SAT?
- Why do we need to solve SAT?

2 Solving SAT
- DPLL
- Parallelization

3 Graphs and SAT
- Graph Representations of SAT Instances
- Connected Components of SAT Formulas

Davis Putnam Logemann Loveland

- A complete deterministic algorithm for SAT solving
- Introduced in 1962, intesively used and improved since 1990
- Basically it is a depth first search of the tree of partial truth assignments with 3 improvements:
  - early termination
  - unit propagation
  - pure literal elimination
- Requires exponential time in the worst case. Very fast for many “real-life” applications.

DPLL Pseudocode

```
function DPLL-SAT($F$): Boolean
    $clauses$ = clausesOf($F$)
    $vars$ = variablesOf($F$)
    $e = \emptyset$ //partial truth assignment
    return DPLL($clauses, vars, e$)

function DPLL($clauses, vars, e$): Boolean
    if $\forall c \in clauses, e^*(c) = true$ then return true
    if $\exists c \in clauses, e^*(c) = false$ then return false
    $e = e \cup$unitPropagation($clauses, e$)
    $e = e \cup$pureLiteralElimination($clauses, e$)
    $x \in vars \wedge x \notin e$ //x is an unassigned variable
    return DPLL($clauses, vars, e \cup -e(x) = true''$) or
        DPLL($clauses, vars, e \cup -e(x) = false''$)
```

Modern improvements of DPLL

- Conflict driven clause learning - new clauses are added to the formula while solving
- Non-chronological backtracking - backtracking more levels at once
- Effective implementation of unit propagation - 2 watched literals scheme
- Restarts - the search is halted and we start from the beginning (the learned clauses are kept)
- Clever decision heuristics

Restarts

- Typically we restart after a given number of conflicts, that number rises exponentially
  - the $n-th$ restart is performed $k.\alpha^{n-1}$ steps after the previous restart. ($k \sim 100$, $\alpha \sim 1.5$)
  - this is called RGR strategy (randomization and geometric restarts)
- Problem sensitive restart heuristics exist
  - observing parameters like length of learned clauses, search depth, backtrack level gives hints about search progress speed
  - computing the moving average of these parameter in an iterative manner
  - examplarily for conflict level, we prefer low values and high variance
  - according to experiments: worse than RGR on satisfiable formulas, but better for unsatisfiable

Conflict Driven Clause Learning DPLL

```
function CDCL-DPLL($F$): Boolean
  $clauses$ =clausesOf($F$)
  $e = \emptyset$ //partial truth assignment
  $level = 0$ //decision level
  if BCP($clauses$,$e$)= $false$ then return false
  while true do
    $lit$ =decide($F$,$e$) //an unassigned variable
    if $lit = null$ then return true
    $level = level + 1$
    $e = e \cup \{e(lit) = true\}$
    while BCP($clauses$,$e$)= $false$ do
       if $level = 0$ return false
       $learned$ =analyzeConflict($clauses$,$e$)
       $clauses = clauses \cup \{learned\}$
       $btLevel$ =computeBTLevel($learned$)
       $e$ =removeLaterAssignments($e$,$btLevel$)
       $level = btLevel$
    endwhile
  endwhile
```

Outline

1 Definition & Motivation
- What is SAT?
- Why do we need to solve SAT?

2 Solving SAT
- DPLL
- Parallelization

3 Graphs and SAT
- Graph Representations of SAT Instances
- Connected Components of SAT Formulas

Parallelization of SAT solving

- This area is not as advanced yet as it should be
- A simple approach is to start the same solver on the same formula multiple times with different parameters
- Another approach is dynamic search space splitting with distributed learning
    - runs of the clause learning DPLL search different parts of the partial assignment tree, but share the learned clauses

Distributed Dynamic Learning (DDL)

- The guiding path is the path in the search tree from the current node to the root
  - each entry has a flag indicating if we are in the first (B) or the second branch (N)
  - each entry with flag B is a potential candidate for search space division (the best candidate is the one closest to the root)
- Search space splitting happens on demand at (almost) any node of the search tree
  - any thread can split itself using its guiding path
  - threads are split when a processor (core) becomes idle
- If any thread solves the whole formula, every thread is stopped and we are finished
- If a thread discovers, that its part of the search tree does not contain any solutions, this thread and all its descendants are terminated

DDL - Learned Clause Sharing

- Each thread has an agent to collect learned clauses
- The agent visits all the threads and collects newly learned clauses if they satisfy the following conditions:
    - the clause is not too long (a constant parameter of the program)
    - the clause is not subsumed by the thread which sent the agent
- The agents run in parallel also
- According to experiments, this kind of parallelization can result in superlinear speedup

Outline

1 Definition & Motivation
- What is SAT?
- Why do we need to solve SAT?

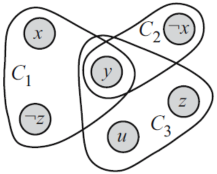
2 Solving SAT
- DPLL
- Parallelization

3 Graphs and SAT
- Graph Representations of SAT Instances
- Connected Components of SAT Formulas

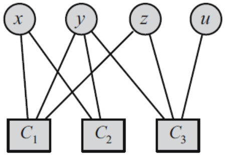
Graph Definitions

- A hypergraph $H = (V, E)$ represents a formula where V is the set of literals and each clause is represented by a hyperedge in E.
- A factor graph $F = (V, E)$ is a bipartite graph, where the variables and clauses are the vertices and there is an edge between the clause and the variable if the clause contains that variable. It can be oriented
- A variable interaction graph $I = (V, E)$ represents a formula with variables V and there is an edge between 2 variables if their literals appear together in a clause
- A resolution graph $R = (C, E)$ is an undirected graph, where the clauses are the vertices and there is an edge between 2 clauses if they can be resolved

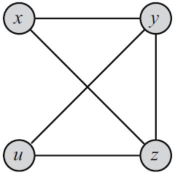
Graph Examples



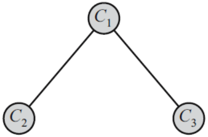
hypergraph



factor graph



variable interaction graph



resolution graph

$$(x \vee y \vee \neg z) \wedge (\neg x \vee y) \wedge (u \vee y \vee z)$$

Usage of Graph Representations

- Visualization of SAT instances can help us to understand why are some instances hard to solve a others not
  - it has been observed, that interaction graphs of hard unsatisfiable formulas have fractal-like patterns
- We can visualize formulas also during the search to understand how DPLL works
  - this can help us to develop better decision heuristics
- We are particularly interested in the connected components of interaction graphs

Outline

1 Definition & Motivation
- What is SAT?
- Why do we need to solve SAT?

2 Solving SAT
- DPLL
- Parallelization

3 Graphs and SAT
- Graph Representations of SAT Instances
- Connected Components of SAT Formulas

Connected Components

- We would like to detect connected components of the interaction graph of a formula and solve the components separately
  - exponential speedup ($2^{100} \rightarrow 2^{50} + 2^{50} = 2^{51} \Rightarrow 2^{49} times\, faster$)
- The input formula has almost allways only one component
  - but it is quickly being disconnected during DPLL search
    - we can remove vertices which have a value in the current partial truth assignment
    - we can remove edges which represent satisfied clauses

Detecting Connected Components

- Detection of components during search is prohibitively expensive (even if we use the best version of the union-find algorithm)
  - for this reason most solvers do not care about connected components
- We can use component friendly decision/phase heuristics (BerkMin, Phase Saving)
  - these heuristics do not explicitly detect connected components, but thanks to their properties they solve them separately and effectively
- COMPSAT uses component detection at the beginning of the search and everytime a unary clause is learned

Summary

- Solving SAT is important
- Yes, it is exponential in the worst case, but works fine most of the practical applications
- A lot of research is being done on SAT solving
- There is still a lot to be improved

For Further Reading I

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (eds.).
Handbook of Satisfiability.
IOS Press, 2009.