Boolean Satisfiability: From Theoretical Hardness to Practical Success

Sharad Malik
Princeton University
SAT in a Nutshell

- Given a Boolean formula, find a variable assignment such that the formula evaluates to 1, or prove that no such assignment exists.

\[ F = (a + b)(a' + b' + c) \]

- For \( n \) variables, there are \( 2^n \) possible truth assignments to be checked.

- First established NP-Complete problem.

Where are we today?

- Intractability of the problem no longer daunting
  - Can regularly handle practical instances with millions of variables and constraints

- SAT has matured from theoretical interest to practical impact
  - Electronic Design Automation (EDA)
    - Widely used in many aspects of chip design
  - Increasing use in software verification
    - Commercial use at Microsoft, NEC,…
Welcome to SAT Live!

If you are a newcomer to the SATisfiability problem, you might want to take a look at wikipedia’s page on the boolean satisfiability problem first. You might also find those survey articles insightful. The current interest in SAT solvers for software and hardware verification, Armin's course on formal methods is a good start. Eugene Goldberg has also written a nice article on practical applications of boolean satisfiability.

Looking for a SAT solver to play with? The following open source SAT solvers might be a good start: MiniSat (C++), Picosat (C), SAT4J (Java). If you are looking for a stochastic local search framework, take a look at UBCSAT.

You can take a look at all the current links, see the links classified by keywords or add your own reference (you must be subscribed to SAT Live! or propse it as anonymous).

If you don’t have some links to propose for now but would like email notification of new additions to the repository, you can subscribe to the SAT Live! notification list or register to the site.

Finally, a page with some people interested in the SATisfaction problem is also available.

Last 10 new entries

| Date:      | 09-Jun-2011 |
| Title:     | Offer for a PhD position or a Post-doc position |
| Hits:      | 18         |
| Contributed by: | Stephan Esdersh@ |
| Keywords:  | Job        |

Innovative approaches to guarantee correctness while designing embedded systems

Location

Group of Computer Architecture headed by Prof. Dr. Rolf Drechsler
University of Bremen, Bremen, Germany

Application

The deadline for applications is July 10th 2011. Applications including CV, certificates, and recommendation letters should be sent by email to Rolf Drechsler (drechsler@uni-bremen.de) for reference number A 53/11.

Salary

Dependent on the qualification of the applicant the salary grade for the position as a researcher (Wissenschaftliche/r MitarbeiterIn) will be TVL 13 or TVL 14, i.e. net income of 1800 EUR or 2000 EUR, respectively. The project will start on August 1st 2011.

Abstract

The internationally renowned Group of Computer Architecture at the University of Bremen develops design automation tools for circuits and systems. Focus of the offered project is the development of innovative approaches to guarantee correctness while designing embedded systems. The position is part of a research project funded by the German Research Foundation for 5 years within a Reinhart Koselleck Project.

The research group tightly cooperates with industrial partners within transfer projects, funded e.g. by the German Ministry for Education and Research (BMBF). Within these...
SAT Competition 2011
A competitive event of the SAT 2011 Conference
June 19th - June 22nd 2011, Ann Arbor, MI, USA


Registration

Register and submit your solver or benchmark

What’s new this year?

There are several new features in the SAT competition this year:

New Hardware
The competition will run on a new cluster at CRIL, composed of nodes with two Intel Xeon Quad core processors and 32 GB of RAM. The operating system is CentOS 5.4, x86. Each instance of a sequential solver will be allocated 7GB of memory. Each instance of a solver will be allocated 2 cores, each instance of a solver will be running either 4 runs of a sequential solver (2 per processor), or 2 runs of a parallel solver (1 per processor). Two different solvers will be used on each processor (2 solvers per node).

Sequential/Parallel Neutrality
This year, there is no special track dedicated to sequential or parallel solvers. Sequential and parallel solvers are grouped into a single competition, but with two different wall clock time and will promote solvers which use all available resources to give an answer as quickly as possible. The second ranking is based on CPU time and will promote most efficiently as possible. This latter ranking is the one that was used in the previous competitions. In the wall clock based ranking, timeout will be imposed on the wall clock time. In will be imposed on CPU time. It is expected that parallel solvers will perform better in the first ranking while sequential solvers will perform better in the second ranking.

New Award Categories
The competition will award both the fastest SAT solvers in terms of wall-clock time and in terms of CPU time. The most innovative (“non CDCL”) SAT solver will be awarded a

Choose Your Category
Unlike the previous competitions, in which all solvers where run on all benchmarks, in order to save computational resources this time submitters are asked to select in which category (random) their solver will compete. The most efficient solvers (selected by the jury) will still compete in every category during the second stage.

Minimally Unsatisfiable Subset (MUS) Special Track
Due to the success of MUS techniques on various applications (especially as core engines in MAXSAT solvers), a special track for MUS systems will be organized for the first time.

Data Analysis Track
Since there are many different ways to analyze the results of the competition, the Data Analysis Track will offer to anyone the possibility to run its own analysis of the competition the competition web site and as a poster during the SAT conference. This track is an opportunity to experiment different ranking schemes, as well as analyze the strengths and weaknesses of the benchmark.

Competition tracks

Here is a quick view of the competition. See detailed rules for complete details.

Main track
Where are we today? (contd.)

- Significant SAT community
  - SatLive Portal and SAT competitions
  - SAT Conference

- Emboldened researchers to take on even harder problems
  - Satisfiability Modulo Theories (SMT)
  - Max-SAT
  - Quantified Boolean Formulas (QBF)
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- **“Pre-processing”**
  - Peephole optimization, e.g. miniSAT, 2005
Problem Representation

- Conjunctive Normal Form
  - Representation of choice for modern SAT solvers

\[(a+b+c)(a'+b'+c)(a'+b+c')(a+b'+c')\]
Circuit to CNF Conversion

- **Tseitin Transformation**

  \[
  d \equiv (a + b) \\
  (a + b + d') \\
  (a' + d) \\
  (b' + d) \\
  \\
  e \equiv (c \cdot d) \\
  (c' + d' + e) \\
  (d + e') \\
  (c + e')
  \]

  Consistency conditions for circuit variables

- **Can ‘e’ ever become true?**

  Is \((e)(a + b + d')(a' + d)(b' + d)(c' + d + e)(d + e')(c + e')\) satisfiable?
Resolution

- Resolution of a pair of distance-one clauses

\[(a + b + c' + f) \land (g + h' + c + f)\]

Resolvent implied by the original clauses

\[a + b + g + h' + f\]

- Iterative existential quantification of variables

Potential memory explosion problem!
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- **“Pre-processing”**
  - Peephole optimization, e.g. miniSAT, 2005
Basic DLL Search

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(b' + c' + d)\]
\[(a' + b + c')\]
\[(a' + b' + c)\]

Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a + c' + d') \\
(b' + c' + d) \\
(a' + b + c') \\
(a' + b' + c)
\end{align*}
\]
Basic DLL Search

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(a' + b + c)\]
\[(a' + b' + c)\]
\[(a' + b' + c')\]

\[\text{Decision} \leftrightarrow 0\]
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a' + b + c) \\
(a' + c' + d) \\
(b' + c' + d) \\
(a' + b + c') \\
(a' + b' + c)
\end{align*}
\]

\[
\begin{align*}
\rightarrow (a + c + d) \\
\rightarrow (a + c' + d')
\end{align*}
\]
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a + c' + d') \\
(a' + b + c) \\
(a' + b' + c) \\
\end{align*}
\]

Unit Clause Rule

Implication Graph

Unit Clause Rule
Basic DLL Search

\begin{align*}
(a' + b + c) & \\
(a + c + d) & \\
(a + c + d') & \\
(a + c' + d) & \\
(a + c' + d') & \\
(a' + b + c) & \\
(a' + b + c') & \\
(a' + b' + c) & \\
\end{align*}

Implication Graph

\begin{itemize}
\item \(a = 0\) \(\rightarrow\) \(d = 1\)
\item \(c = 0\) \(\rightarrow\) \(d = 0\)
\end{itemize}
Basic DLL Search

\[(a' + b + c)\]
\[\rightarrow\]
\[a + c + d\]
\[\rightarrow\]
\[a + c + d'\]

\[a' + b + c\]
\[\rightarrow\]
\[b' + c' + d\]
\[\rightarrow\]
\[a' + b + c'\]
\[\rightarrow\]
\[a' + b' + c\]

Conflict! Implication Graph

\[a = 0\]
\[\rightarrow\]
\[d = 1\]
\[(a + c + d)\]

\[c = 0\]
\[\rightarrow\]
\[d = 0\]
\[(a + c + d')\]

\[d = 1, d = 0\]
Basic DLL Search

\[(a' + b + c)\]
\[\rightarrow (a + c + d)\]
\[\rightarrow (a + c + d')\]
\[\rightarrow (a + c' + d)\]
\[\rightarrow (a + c' + d')\]
\[\rightarrow (b' + c' + d)\]
\[\rightarrow (a' + b + c')\]
\[\rightarrow (a' + b' + c)\]

\[\xleftarrow{\text{Backtrack}}\]
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a' + b + c) \\
(b' + c' + d) \\
(a' + b + c') \\
(a' + b' + c)
\end{align*}
\]

\[
\rightarrow 
\rightarrow
\]

\[
\begin{align*}
(a + c' + d) \\
(a + c' + d')
\end{align*}
\]

\[
\rightarrow 
\rightarrow
\]

\[
\begin{align*}
(b' + c' + d) \\
(a' + b + c')
\end{align*}
\]

\[
\text{Forced Decision}
\]
Basic DLL Search

Implication Graph

Conflict!
Basic DLL Search

\[(a' + b + c)\]
\rightarrow \[(a + c + d)\]
\rightarrow \[(a + c + d')\]
\rightarrow \[(a + c' + d)\]
\rightarrow \[(a + c' + d')\]
\rightarrow \[(b' + c' + d)\]
\rightarrow \[(a' + b + c')\]
\rightarrow \[(a' + b' + c)\]
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

Backtrack
Basic DLL Search

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(b' + c' + d)\]
\[(a' + b + c')\]
\[(a' + b' + c)\]
Basic DLL Search

\[(a' + b + c)
\]

⇒

\[(a + c + d)
\]

⇒

\[(a + c + d')
\]

⇒

\[(a + c' + d)
\]

⇒

\[(a + c' + d')
\]

⇒

\[(a' + b + c)
\]

⇒

\[(b' + c' + d)
\]

⇒

\[(a' + b + c')
\]

⇒

\[(a' + b' + c)
\]

Conflict!

\[d=1, d=0\]

Decision

Implication Graph

Conflict!
Basic DLL Search

\[(a' + b + c)\]  
\[\rightarrow\]  
\[(a + c + d)\]  
\[\rightarrow\]  
\[(a + c + d')\]  
\[\rightarrow\]  
\[(a + c' + d)\]  
\[\rightarrow\]  
\[(a + c' + d')\]  
\[\rightarrow\]  
\[(b' + c' + d)\]  
\[\rightarrow\]  
\[(a' + b + c')\]  
\[\rightarrow\]  
\[(a' + b' + c)\]  

\[\leftarrow\text{Backtrack}\]
Basic DLL Search

\[ (a' + b + c) \]
\[ (a + c + d) \]
\[ (a + c + d') \]
\[ (a' + b + c') \]
\[ (a' + b' + c) \]
\[ \]
Basic DLL Search

\[
\begin{align*}
(a' + b + c) \\
(a + c + d) \\
(a + c + d') \\
(a + c' + d) \\
(a + c' + d') \\
(b' + c' + d) \\
(a' + b + c') \\
(a' + b' + c)
\end{align*}
\]
Basic DLL Search

\[(a' + b + c)\]
\[(a + c + d)\]
\[(a + c + d')\]
\[(a + c' + d)\]
\[(a + c' + d')\]
\[(a' + b + c)\]
\[(b' + c' + d)\]
\[(a' + b + c')\]
\[(a' + b' + c)\]

\[\text{Forced Decision}\]
Basic DLL Search

\( (a' + b + c) \)
\( (a + c + d) \)
\( (a + c + d') \)
\( (a + c' + d) \)
\( (a + c' + d') \)
\( (b' + c' + d) \)
\( (a' + b + c') \)
\( (a' + b' + c) \)

\( a=1 \)
\( b=1 \)
\( c=1 \)

Implication Graph
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a + c' + d)
(a + c' + d')
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

Implication Graph

(a' + b' + c)
(b' + c' + d)
(a' + b' + c)

a=1
c=1
b=1
d=1

C=1, d=1
Basic DLL Search

(a' + b + c)
(a + c + d)
(a + c + d')
(a' + c' + d)
(b' + c' + d)
(a' + b + c')
(a' + b' + c)

(b' + c' + d)

(a' + b' + c)
(c' + d' + c)
(d' + c' + d)

Implication Graph

a=1
b=1
c=1
d=1

a

b

c

b

(c=1, d=1)

SAT
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- “Pre-processing”
  - Peephole optimization, e.g. miniSAT, 2005
Conflict Driven Learning and Non-chronological Backtracking

\[
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_{12} \\
x_2 + x_{11} \\
x_{7'} + x_3' + x_9 \\
x_{7'} + x_8 + x_9' \\
x_7 + x_8 + x_{10'} \\
x_7 + x_{10} + x_{12'}
\]

Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_{7'} + x_3' + x_9 \]
\[ x_{7'} + x_8 + x_9' \]
\[ x_9 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

\[ x_1 = 0 \]
Conflict Driven Learning and Non-chronological Backtracking

\begin{align*}
    x_1 + x_4 \\
    x_1 + x_3' + x_8' \\
    x_1 + x_8 + x_{12} \\
    x_2 + x_{11} \\
    x_7' + x_3' + x_9 \\
    x_7' + x_8 + x_9' \\
    x_7 + x_8 + x_{10'} \\
    x_7 + x_{10} + x_{12'} \\
\end{align*}

\begin{itemize}
    \item $x_1 = 0, \ x_4 = 1$
    \item $x_1 = 0$
    \item $x_4 = 1$
\end{itemize}
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_12 \]
\[ x_2 + x_11 \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_10' \]
\[ x_7 + x_10 + x_12' \]

\[ x_4 = 1 \]
\[ x_1 = 0, x_3 = 1 \]
\[ x_1 = 0, x_4 = 1 \]
\[ x_3 = 1 \]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
Conflict Driven Learning and Non-chronological Backtracking

\begin{align*}
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_{12} \\
x_2 + x_{11} \\
x_7' + x_3' + x_9 \\
x_7' + x_8 + x_9' \\
x_7 + x_8 + x_{10'} \\
x_7 + x_{10} + x_{12'}
\end{align*}
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10}' \]
\[ x_7 + x_{10} + x_{12}' \]
Conflict Driven Learning and Non-chronological Backtracking

\[
\begin{align*}
&x_1 + x_4 \\
&x_1 + x_3' + x_8' \\
&x_1 + x_8 + x_{12} \\
&x_2 + x_{11} \\
&x_7' + x_3' + x_9 \\
&x_7' + x_8 + x_9' \\
&x_7 + x_8 + x_{10'} \\
&x_7 + x_{10} + x_{12'} \\
\end{align*}
\]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]
Conflict Driven Learning and Non-chronological Backtracking

\[ \begin{align*}
&x_1 + x_4 \\
&x_1 + x_3' + x_8' \\
&x_1 + x_8 + x_{12} \\
&x_2 + x_{11} \\
&x_7' + x_3' + x_9 \\
&x_7' + x_8 + x_9' \\
&x_7 + x_8 + x_{10'} \\
&x_7 + x_{10} + x_{12'}
\end{align*} \]

\[\begin{align*}
x_1 &= 0, \quad x_4 = 1 \\
x_3 &= 1, \quad x_8 = 0, \quad x_{12} = 1 \\
x_2 &= 0, \quad x_{11} = 1 \\
x_7 &= 1, \quad x_9 = 1 \\
x_4 &= 1 \\
x_9 &= 0 \\
x_1 &= 0 \\
x_2 &= 0 \\
x_{11} &= 1 \\
x_{12} &= 1
\end{align*}\]

\[x_3 = 1 \land x_7 = 1 \land x_8 = 0 \rightarrow \text{conflict}\]
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_7' + x_3' + x_9 \]
\[ x_7' + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

Add conflict clause: \( x_3' + x_7' + x_8 \)

\[ x_3 = 1 \land x_7 = 1 \land x_8 = 0 \rightarrow \text{conflict} \]

Add conflict clause: \( x_3' + x_7' + x_8 \)
Conflict Driven Learning and Non-chronological Backtracking

\[ x_1 + x_4 \]
\[ x_1 + x_3' + x_8' \]
\[ x_1 + x_8 + x_{12} \]
\[ x_2 + x_{11} \]
\[ x_{7'} + x_3' + x_9 \]
\[ x_{7'} + x_8 + x_9' \]
\[ x_7 + x_8 + x_{10'} \]
\[ x_7 + x_{10} + x_{12'} \]

\[ x_1 = 0, x_4 = 1 \]
\[ x_3 = 1, x_8 = 0, x_{12} = 1 \]
\[ x_2 = 0, x_{11} = 1 \]
\[ x_7 = 1, x_9 = 1 \]
\[ x_3' + x_{7'} + x_8 \]

Add conflict clause: \[ x_3' + x_{7'} + x_8 \]

\[ x_3 = 1 \land x_7 = 1 \land x_8 = 0 \rightarrow \text{conflict} \]

Add conflict clause: \[ x_3' + x_{7'} + x_8 \]
Conflict Driven Learning and Non-chronological Backtracking

$x1 + x4$
$x1 + x3' + x8'$
$x1 + x8 + x12$
$x2 + x11$
$x7' + x3' + x9$
$x7' + x8 + x9'$
$x7 + x8 + x10'$
$x7 + x10 + x12'$

Backtrack to the decision level of $x3=1$
Conflicts Driven Learning and Non-chronological Backtracking

\[
x_1 + x_4 \\
x_1 + x_3' + x_8' \\
x_1 + x_8 + x_12 \\
x_2 + x_11 \\
x_7' + x_3' + x_9 \\
x_7' + x_8 + x_9' \\
x_7 + x_8 + x_{10}' \\
x_7 + x_{10} + x_{12}' \\
x_3' + x_7' + x_8 
\]

\[\text{← new clause}\]

Backtrack to the decision level of \(x_3=1\)
Assign \(x_7 = 0\)
What’s the big deal?

Conflict clause: $x_1'+x_3+x_5'$

Significantly prune the search space – learned clause is useful forever!

Useful in generating future conflict clauses.
Restart

- Abandon the current search tree and reconstruct a new one.
- The clauses learned prior to the restart are still there after the restart and can help pruning the search space.
- Adds to robustness in the solver.
SAT Solvers: A Condensed History

- **Deductive**
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”

- **Backtrack Search**
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment

- **Conflict Driven Clause Learning [CDCL]**
  - GRASP: Integrate a constraint learning procedure, 1996

- **Locality Based Search**
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation

- **“Pre-processing”**
  - Peephole optimization, e.g. miniSAT, 2005
Success with Chaff

- First major instance: Tough (Industrial Processor Verification)
  - Bounded Model Checking, 14 cycle behavior

- Statistics
  - 1 million variables
  - 10 million literals initially
    - 200 million literals including added clauses
    - 30 million literals finally
  - 4 million clauses (initially)
    - 200K clauses added
  - 1.5 million decisions
  - 3 hour run time

Chaff Contribution 1: Lazy Data Structures
2 Literal Watching for Unit-Propagation

- Avoid expensive book-keeping for unit-propagation
- N-literal clause can be unit or conflicting only after N-1 of the literals have been assigned to F
  - \((v_1 + v_2 + v_3)\): implied cases: \((0 + 0 + v_3)\) or \((0 + v_2 + 0)\) or \((v_1 + 0 + 0)\)
- Can completely ignore the first N-2 assignments to this clause
- Pick two literals in each clause to “watch” and thus can ignore any assignments to the other literals in the clause.
  - Example: \((v_1 + v_2 + v_3 + v_4 + v_5)\)
  - \((v_1=X + v_2=X + v_3=? \text{ i.e. } X \text{ or } 0 \text{ or } 1) + v_4=? + v_5=?\)
- *Maintain the invariant*: If a clause can become newly implied via any sequence of assignments, then this sequence will include an assignment of one of the watched literals to F
When a variable is assigned true, only need to visit clauses where its watched literal is false (only one polarity)

- Pointers from each literal to all clauses it is watched in

In a $n$ clause formula with $v$ variables and $m$ literals

- Total number of pointers is $2n$
- On average, visit $n/v$ clauses per assignment

*No updates to watched literals on backtrack*

For every clause, two literals are watched
**Decision Heuristics — Conventional Wisdom**

- “Assign most tightly constrained variable”: e.g. DLIS (Dynamic Largest Individual Sum)
  - Simple and intuitive: At each decision simply choose the assignment that satisfies the most unsatisfied clauses.
  - Expensive book-keeping operations required
    - Must touch *every* clause that contains a literal that has been set to true.
      - Often restricted to initial (not learned) clauses.
    - Need to reverse the process for un-assignment.
- Look ahead algorithms even more compute intensive
- Take a more “global” view of the problem
Chaff Contribution 2: Activity Based Decision Heuristics

- **VSIDS: Variable State Independent Decaying Sum**
  - Rank variables by literal count in the initial clause database
  - Only increment counts as new (learnt) clauses are added
  - Periodically, divide all counts by a constant

- **Quasi-static:**
  - Static because it doesn’t depend on variable state
  - Not static because it gradually changes as new clauses are added
  - Decay causes bias toward *recent* conflicts.
  - Has a beneficial interaction with 2-literal watching
Activity Based Heuristics
and Locality Based Search

- By focusing on a sub-space, the covered spaces tend to coalesce
  - More opportunities for resolution since most of the variables are common.
  - Variable activity based heuristics lead to locality based search
SAT Solvers: A Condensed History

- Deductive
  - Davis-Putnam 1960 [DP]
  - Iterative existential quantification by “resolution”
- Backtrack Search
  - Davis, Logemann and Loveland 1962 [DLL]
  - Exhaustive search for satisfying assignment
- Conflict Driven Clause Learning [CDCL]
  - GRASP: Integrate a constraint learning procedure, 1996
- Locality Based Search
  - Emphasis on exhausting local sub-spaces, e.g. Chaff, Berkmin, miniSAT and others, 2001 onwards
  - Added focus on efficient implementation
- “Pre-processing”
  - Peephole optimization, e.g. miniSAT, 2005
Pre-Processing of CNF Formulas

N. Eén and A. Biere. Effective Preprocessing in SAT through Variable and Clause Elimination, In Proceedings of SAT 2005

- Use structural information to simplify
  - Subsumption
  - Self-subsumption
  - Substitution
Pre-Processing: Subsumption

- Clause $C_1$ subsumes clause $C_2$ if $C_1$ implies $C_2$
- Subsumed clauses can be discarded

\[(\bar{x} + y) \cdot (\bar{x} + y - z) \cdot (\bar{y} + u) \cdot (\bar{x} + y + z + v) \cdot (\bar{y})\]
Pre-Processing: Self-Subsumption

- Subsumption after resolution step

\[
(\overline{x} + y + z) \quad (x + y + z + \overline{u})
\]

\[
(y + z + \overline{u})
\]
Pre-Processing: Substitution

- Tseitin transformation introduces definition of variable

\[
\begin{align*}
    & (x_1 \leftrightarrow (y \leftrightarrow z)) \\
    & (\overline{x_1} + \overline{y} + z) \cdot (\overline{x_1} + \overline{z} + y) \cdot (\overline{y} + \overline{z} + x_1) \cdot (y + z + x_1)
\end{align*}
\]

- Occurrence of \( x_1 \) can be eliminated by substitution
  - Corresponds to resolution with defining clauses

\[
\begin{align*}
    & (x_1 + u) \cdot (\overline{x_1}) \cdot (y \leftrightarrow z) \cdot (\overline{x_1} + \overline{z} + y) \\
    & (u + \overline{y} + z) \quad (u + x_1) \quad (u + \overline{z} + y)
\end{align*}
\]
Concluding Remarks

- SAT: Significant shift from theoretical interest to practical impact.
- Quantum leaps between generations of SAT solvers
- Successful application of diverse CS techniques
  - Logic (Deduction and Solving), Search, Caching, Randomization, Data structures, efficient algorithms
  - Engineering developments through experimental computer science
- Presence of drivers results in maximum progress.
  - Electronic design automation – primary driver and main beneficiary
  - Software verification- the next frontier
- Opens attack on even harder problems
  - SMT, Max-SAT, QBF…

References

References


References

- [EB05] N. Eén and A. Biere. Effective Preprocessing in SAT through Variable and Clause Elimination, In *Proceedings of SAT 2005*
- [HJS08] Youssef Hamadi, Said Jabbour, and Lakhdar Sais, ManySat: solver description, Microsoft Research-TR-2008-83