Heuristics for Efficient SAT Solving

As implemented in GRASP, Chaff and GSAT.
The $K$-Coloring problem:
Given an undirected graph $G(V,E)$ and a natural number $k$, is there an assignment $\text{color}$:

\[ V \rightarrow \{1, \ldots, k\} \text{ s.t. } \forall i, j, (i, j) \in E, \text{color}(i) \neq \text{color}(j) \]
Formulation of famous problems as SAT: $k$-Coloring (2/2)

$x_{i,j} =$ node $i$ is assigned the ‘color’ $j$ ($1 \leq i \leq n$, $1 \leq j \leq k$)

**Constraints:**

i) At least one color to each node: $(x_{1,1} \lor x_{1,2} \lor \ldots x_{1,k} \lor \ldots)$

\[
\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k} (x_{i,j})
\]

ii) At most one color to each node:

\[
\forall \forall \forall (\neg x_{i,j} \lor \neg x_{i,t})
\]

iii) Coloring constraints:

\[
\forall \forall \forall (i, j) \in E. (\neg x_{i,c} \lor \neg x_{j,c})
\]
Given a property $p$: (e.g. “always signal_a = signal_b”)

Is there a state reachable within $k$ cycles, which satisfies $\neg p$ ?
The reachable states in $k$ steps are captured by:

$$I(s_0) \land \rho(s_0, s_1) \land \rho(s_1, s_2) \land \ldots \land \rho(s_{k-1}, s_k)$$

The property $p$ fails in one of the cycles $1..k$:

$$\neg p_1 \lor \neg p_2 \lor \ldots \lor \neg p_k$$
The safety property $p$ is valid up to cycle $k$ iff $\Omega(k)$ is unsatisfiable:

$$\Omega(k): \quad I_0 \land \bigwedge_{i=0}^{k-1} \rho(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg p_i$$

Formulation of famous problems as SAT: 
*Bounded Model Checking (3/4)*
Formulation of famous problems as SAT: Bounded Model Checking (4/4)

Example: a two bit counter

Initial state: $I_0: \neg l \land \neg r$

Transition: $\rho: l' = (l \neq r), r' = \neg r$

Property: always $(\neg l \lor \neg r)$.

For $k = 2$:

$\varphi: (\neg l_0 \land \neg r_0) \land \begin{align*}
\ell_1 &= (l_0 \neq r_0) \land r_1 = \neg r_0 \\
\ell_2 &= (l_1 \neq r_1) \land r_2 = \neg r_1
\end{align*}$

For $k = 2$, $\Omega(k)$ is unsatisfiable. For $k = 4$ $\Omega(k)$ is satisfiable
What is SAT?

Given a propositional formula in CNF, find an assignment to Boolean variables that makes the formula true:

\[ \omega_1 = (x_2 \lor x_3) \]
\[ \omega_2 = (\neg x_1 \lor \neg x_4) \]
\[ \omega_3 = (\neg x_2 \lor x_4) \]

\[ A = \{ x_1=0, x_2=1, x_3=0, x_4=1 \} \]
Why SAT?

- Fundamental problem from theoretical point of view
- Numerous applications:
  - CAD, VLSI
  - Optimization
  - Bounded Model Checking and other type of formal verification
  - AI, planning, automated deduction
A Basic SAT algorithm

Given $\varphi$ in CNF: $(x,y,z),(-x,y),(-y,z),(-x,-y,-z)$

Decision tree:
- $x$: $(y),(-y,z),(-y,-z)$
  - $y$: $(z),(-z)$
  - $z$: $(y),(-y)$

- $-x$: $(y,z),(-y,z)$
  - $y$: $(z),(-z)$
  - $z$: $(y),(-y)$

Resolve Conflict:
- $x$: $y$ is decided as $1$
- $-x$: $y$ is decided as $0$
A Basic SAT algorithm

While (true)
{
    if (!Decide()) return (SAT);
    while (!Deduce())
        if (!Resolve_Conflict()) return (UNSAT);
}

Choose the next variable and value. Return False if all variables are assigned

Apply unit clause rule. Return False if reached a conflict

Backtrack until no conflict. Return False if impossible
Basic Backtracking Search

- Organize the search in the form of a decision tree
  - Each node corresponds to a decision
  - Depth of the node in the decision tree $\rightarrow$ decision level
  - Notation: $x=\nu@d$
    $x \in \{0,1\}$ is assigned to $\nu$ at decision level $d$
Backtracking Search in Action

\[ \omega_1 = (x_2 \lor x_3) \]
\[ \omega_2 = (\neg x_1 \lor \neg x_4) \]
\[ \omega_3 = (\neg x_2 \lor x_4) \]

\[ x_1 = 0 \iff 1 \]
\[ x_2 = 0 \iff 2 \]
\[ x_3 = 1 \iff 2 \]
\[ x_4 = 0 \iff 1 \]

\{ (x_1, 0), (x_2, 0), (x_3, 1) \}

\{ (x_1, 1), (x_2, 0), (x_3, 1), (x_4, 0) \}

No backtrack in this example!
Backtracking Search in Action

Add a clause

\( \omega_1 = (x_2 \lor x_3) \)
\( \omega_2 = (\neg x_1 \lor \neg x_4) \)
\( \omega_3 = (\neg x_2 \lor x_4) \)
\( \omega_4 = (\neg x_1 \lor x_2 \lor \neg x_3) \)

\( x_2 = 0 @ 2 \Rightarrow x_3 = 1 @ 2 \)

\{(x_1,0), (x_2,0), (x_3,1)\}

\begin{align*}
x_1 &= 1 @ 1 \\
\Rightarrow x_4 &= 0 @ 1 \\
\Rightarrow x_2 &= 0 @ 1 \\
\Rightarrow x_3 &= 1 @ 1
\end{align*}
**Decision heuristics**

- **DLIS** (Dynamic Largest Individual Sum)
  - For a given variable $x$:
    - $C_{x,p}$ – # unresolved clauses in which $x$ appears positively
    - $C_{x,n}$ – # unresolved clauses in which $x$ appears negatively
    - Let $x$ be the literal for which $C_{x,p}$ is maximal
    - Let $y$ be the literal for which $C_{y,n}$ is maximal
    - If $C_{x,p} > C_{y,n}$ choose $x$ and assign it TRUE
    - Otherwise choose $y$ and assign it FALSE

- Requires $l$ (#literals) queries for each decision.

- (Implemented in e.g. Grasp)
Decision heuristics

Jeroslow-Wang method

Compute for every clause \( \omega \) and every variable \( l \) (in each phase):

- \( J(l) := \sum_{l \in \omega, \omega \in \varphi} 2^{-|\omega|} \)

- Choose a variable \( l \) that maximizes \( J(l) \).

- This gives an exponentially higher weight to literals in shorter clauses.
Decision heuristics

MOM (Maximum Occurrence of clauses of Minimum size).

- Let $f^*(x)$ be the # of unresolved smallest clauses containing $x$. Choose $x$ that maximizes:
  \[
  ((f^*(x) + f^!(x)) \times 2^k + f^*(x) \times f^!(x)
  \]

- $k$ is chosen heuristically.

- The idea:
  - Give preference to satisfying small clauses.
  - Among those, give preference to balanced variables (e.g. $f^*(x) = 3$, $f^!(x) = 3$ is better than $f^*(x) = 1$, $f^!(x) = 5$).
Decision heuristics

**VSIDS** (Variable State Independent Decaying Sum)

1. Each variable in each polarity has a counter initialized to 0.
2. When a clause is added, the counters are updated.
3. The unassigned variable with the highest counter is chosen.
4. Periodically, all the counters are divided by a constant.

(Implemented in **Chaff**)
Decision heuristics

VSIDS (cont’d)

• **Chaff** holds a list of unassigned variables sorted by the counter value.

• Updates are needed only when adding conflict clauses.

• Thus - decision is made in constant time.
Decision heuristics

VSIDS is a ‘quasi-static’ strategy:

- static because it doesn’t depend on current assignment

- dynamic because it gradually changes. Variables that appear in recent conflicts have higher priority.

This strategy is a conflict-driven decision strategy.

“..employing this strategy dramatically (i.e. an order of magnitude) improved performance ... “
Variable ordering

(Abstract dependency graphs)

A (CNF) dependency graph $D (V,E)$:

A partitioning $C_1..C_n$:

An *abstract* dependency graph $D'(V', E')$: 
Variable ordering
(The natural order of $\Omega(k)$)

For $\Omega(k)$ there exists a partition $C_1..C_n$ s.t. the abstract dependency graph is linear

$$\Omega(k) : \quad I(s_0) \land \bigwedge_{i=1}^{k-1} \rho(s_i, s_{i+1}) \land \neg p(s_k)$$
Variable ordering

(\(\Omega(k)\) should satisfy \(I_0\))

Riding on unreachable states...

\(\neg P_k\)

Riding on legal executions...

\(\Omega(k)\) should satisfy \(\neg P_k\)
Implication graphs and learning

Current truth assignment: \{x_9 = 0 @ 1, x_{10} = 0 @ 3, x_{11} = 0 @ 3, x_{12} = 1 @ 2, x_{13} = 1 @ 2\}

Current decision assignment: \{x_1 = 1 @ 6\}

\begin{align*}
\omega_1 &= \neg x_1 \lor x_2 \\
\omega_2 &= \neg x_1 \lor x_3 \lor x_9 \\
\omega_3 &= \neg x_2 \lor \neg x_3 \lor x_4 \\
\omega_4 &= \neg x_4 \lor x_5 \lor x_{10} \\
\omega_5 &= \neg x_4 \lor x_6 \lor x_{11} \\
\omega_6 &= \neg x_5 \lor \neg x_6 \\
\omega_7 &= x_1 \lor x_7 \lor \neg x_{12} \\
\omega_8 &= x_1 \lor x_8 \\
\omega_9 &= \neg x_7 \lor \neg x_8 \lor \neg x_{13}
\end{align*}

We learn the conflict clause \(\omega_{10} : (\neg x_1 \lor x_9 \lor x_{11} \lor x_{10})\)
Implication graph, flipped assignment

\( \omega_1 = (\neg x_1 \lor x_2) \)
\( \omega_2 = (\neg x_1 \lor x_3 \lor x_9) \)
\( \omega_3 = (\neg x_2 \lor \neg x_3 \lor x_4) \)
\( \omega_4 = (\neg x_4 \lor x_5 \lor x_{10}) \)
\( \omega_5 = (\neg x_4 \lor x_6 \lor x_{11}) \)
\( \omega_6 = (\neg x_5 \lor x_6) \)
\( \omega_7 = (x_1 \lor x_7 \lor \neg x_{12}) \)
\( \omega_8 = (x_1 \lor x_8) \)
\( \omega_9 = (\neg x_7 \lor \neg x_8 \lor \neg x_{13}) \)
\( \omega_{10} : (\neg x_1 \lor x_9 \lor x_{11} \lor x_{10}) \)
Non-chronological backtracking

Which assignments caused the conflicts?

\[
\begin{align*}
  x_9 &= 0 @ 1 \\
  x_{10} &= 0 @ 3 \\
  x_{11} &= 0 @ 3 \\
  x_{12} &= 1 @ 2 \\
  x_{13} &= 1 @ 2
\end{align*}
\]

These assignments are sufficient for causing a conflict.

Backtrack to decision level 3
More engineering aspects of SAT solvers

Observation: More than 90% of the time SAT solvers perform Deduction().

Deduction() allocates new implied variables and conflicts. How can this be done efficiently?
Grasp implements Deduction() with counters

Hold 2 counters for each clause $\pi$:

$val1(\pi)$ - \# of \underline{negative} literals assigned 0 in $\pi$ + \# of \underline{positive} literals assigned 1 in $\pi$.

$val0(\pi)$ - \# of \underline{negative} literals assigned 1 in $\pi$ + \# of \underline{positive} literals assigned 0 in $\pi$. 
**Grasp** implements Deduction() with counters

\[
\begin{align*}
\pi \text{ is satisfied} & \quad \text{iff} \quad \text{val}_1(\pi) > 0 \\
\pi \text{ is unsatisfied} & \quad \text{iff} \quad \text{val}_0(\pi) = |\pi| \\
\pi \text{ is unit} & \quad \text{iff} \quad \text{val}_1(\pi) = 0 \land \text{val}_0(\pi) = |\pi| - 1 \\
\pi \text{ is unresolved} & \quad \text{iff} \quad \text{val}_1(\pi) = 0 \land \text{val}_0(\pi) < |\pi| - 1 \\
\end{align*}
\]

Every assignment to a variable \(x\) results in updating the counters for all the clauses that contain \(x\).

**Backtracking:** Same complexity.
Chaff implements Deduction() with a pair of observers

- Observation: during Deduction(), we are only interested in newly implied variables and conflicts.
- These occur only when the number of literals in $\pi$ with value ‘false’ is greater than $|\pi| - 2$
- Conclusion: no need to visit a clause unless $(\text{val0} (\pi) > |\pi| - 2)$

- How can this be implemented?
Chaff implements Deduction() with a pair of observers

- Define two ‘observers’: $O1(\pi)$, $O2(\pi)$.
- $O1(\pi)$ and $O2(\pi)$ point to two distinct $\pi$ literals which are not ‘false’.
- $\pi$ becomes *unit* if updating one observer leads to $O1(\pi) = O2(\pi)$.
- Visit clause $\pi$ only if $O1(\pi)$ or $O2(\pi)$ become ‘false’.
Both observers of an implied clause are on the highest decision level present in the clause. Therefore, backtracking will un-assign them first. Conclusion: when backtracking, observers stay in place.

Backtracking: No updating. Complexity = constant.
Chaff implements Deduction() with a pair of observers

The choice of observing literals is important.

Best strategy is - the least frequently updated variables.

The observers method has a learning curve in this respect:

1. The initial observers are chosen arbitrarily.

2. The process shifts the observers away from variables that were recently updated (these variables will most probably be reassigned in a short time).

In our example: the next time v[5] is updated, it will point to a significantly smaller set of clauses.
Given a CNF formula $\alpha$, choose \texttt{max\_tries} and \texttt{max\_flips}

\begin{verbatim}
for i = 1 to \texttt{max\_tries} {
    T := randomly generated truth assignment
    for j = 1 to \texttt{max\_flips} {
        if T satisfies $\alpha$ return TRUE
        choose v s.t. flipping v's value gives largest increase in
            the \# of satisfied clauses (break ties randomly).
        T := T with v's assignment flipped. }
    }
\end{verbatim}
Improvement # 1: clause weights

Initial weight of each clause: 1

Increase by $k$ the weight of unsatisfied clauses.

Choose $v$ according to max increase in weight

Clause weights is another example of conflict-driven decision strategy.
Improvement # 2: Averaging-in

Q: Can we reuse information gathered in previous tries in order to speed up the search?

A: Yes! Rather than choosing T randomly each time, repeat ‘good assignments’ and choose randomly the rest.
Let $X_1$, $X_2$ and $X_3$ be equally wide bit vectors.

Define a function \texttt{bit\_average} : $X_1 \times X_2 \rightarrow X_3$ as follows:

\[
    b_3^i := \begin{cases} 
        b_1^i & \text{if } b_1^i = b_2^i \\
        \text{random} & \text{otherwise}
    \end{cases}
\]

(where $b_j^i$ is the $i$-th bit in $X_j$, $j \in \{1,2,3\}$)
**Improvement # 2: Averaging-in (cont’d)**

Let $T_i^{\text{init}}$ be the initial assignment ($T$) in cycle $i$.
Let $T_i^{\text{best}}$ be the assignment with highest # of satisfied clauses in cycle $i$.

- $T_1^{\text{init}} := \text{random assignment}$.
- $T_2^{\text{init}} := \text{random assignment}$.
- $\forall i > 2, T_i^{\text{init}} := \text{bit\_average}(T_{i-1}^{\text{best}}, T_{i-2}^{\text{best}})$