Overview

Introduction

Tractable cases

DPLL algorithms
- Monien-Speckenmeyer algorithm
- Kullmann’s method
- Algorithm of Zhang

CDCL solvers

Lookahead-based solvers

Probabilistic algorithms

Certification

Applications
The basic DPLL algorithm

After Davis, Putnam, Logemann and Loveland, 1960

The basic DPLL Algorithm:

DPLL($F, \alpha$)

- if $F\alpha = 0$ then return UNSAT
- if $F\alpha = 1$ then return $\alpha$

pick $x \in V(F\alpha)$

$\beta := $ DPLL($F, \alpha \cup [x := 0]$)

if $\beta \neq$ UNSAT then return $\beta$
else return DPLL($F, \alpha \cup [x := 1]$)
A simple analysis

For a $k$-CNF formula $F$, iterate the following:

- pick a clause $C$ in $F$
- branch successively on the $k$ variables in $V(C)$

Of the $2^k$ assignments to $V(C)$, one sets $\alpha(C) = \alpha(F) = 0$.

Thus: $2^k - 1$ branching tree of height $n/k$.

Runtime is essentially the tree-size $b_k^n$ for $b_k := \sqrt[k]{2^k - 1} < 2$

<table>
<thead>
<tr>
<th>$k$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_k$</td>
<td>1.91294</td>
<td>1.96799</td>
<td>1.98735</td>
<td>1.99477</td>
<td>1.99777</td>
<td>1.99903</td>
</tr>
</tbody>
</table>
Unit propagation

A branching strategy (almost) always employed:
  ▶ pick $x$ if $x$ occurs in a unit clause $a$.

One branch immediately fails
$\leadsto$ just set $[a := 1]$ instead of branching

Mostly realized as simplification step before branching:

$$\text{UnitProp}(F, \alpha)$$

while $F\alpha$ contains unit clause $a$

$$\alpha := \alpha \cup [a := 1]$$

return $\alpha$
Pure literals and subsumption

Other simplification steps in original DPLL:

- elimination of pure literals
- deletion of subsumed clauses

\text{PureLit}(F, \alpha)

\text{while } F\alpha \text{ contains pure literal } a
\[\alpha := \alpha \cup [a := 1]\]

Clause \( C \) subsumes \( D \) if \( C \subseteq D \)

\text{Subs}(F)

\text{while } F \text{ contains clauses } C \subseteq D
\[F := F \setminus D\]
The general DPLL algorithm

\[ \text{DPLL}(F, \alpha) \]

\[ \text{simplify}(F, \alpha) \]

if $F = 0$ then return UNSAT
if $F = 1$ then return $\alpha$

pick $x \in V(F)$ and $\epsilon \in \{0, 1\}$

$\beta := \text{DPLL}(F[x := \epsilon], \alpha \cup [x := \epsilon])$

if $\beta \neq \text{UNSAT}$
    then return $\beta$
else return $\text{DPLL}(F[x := \bar{\epsilon}], \alpha \cup [x := \bar{\epsilon}])$
Monien-Speckenmeyer algorithm - simple version

Branching strategy:
  ▶ pick literal from a shortest clause

Equivalently:

\[
simpleMS(F, \alpha)
\]

\[
\text{if } F\alpha = 0 \text{ then return UNSAT}
\]

\[
\text{if } F\alpha = 1 \text{ then return } \alpha
\]

pick shortest clause \( C = a_1 \lor \ldots \lor a_r \)

\[
\text{for } i := 1 \text{ to } r \text{ do}
\]

\[
\gamma_i := [a_1 := 0, \ldots, a_{i-1} := 0, a_i := 1]
\]

\[
\beta := simpleMS(F, \alpha \cup \gamma_i)
\]

\[
\text{if } \beta \neq \text{ UNSAT } \text{ then return } \beta
\]

return UNSAT
Analysis of simpleMS

At each node: $r$-fold branching for some $r \leq k$, in $i$th branch $n - i$ variables.

Recursion for the tree-size:

$$T(n) := T(n - 1) + \ldots + T(n - k)$$

Set $T(n) := b^n$, thus the basis $b_k = b$ satisfies

$$b^k := b^{k-1} + \ldots + b + 1$$

The solutions for small $k$ are:

<table>
<thead>
<tr>
<th>$k$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_k$</td>
<td>1.83929</td>
<td>1.92757</td>
<td>1.96595</td>
<td>1.98359</td>
<td>1.99197</td>
<td>1.99603</td>
</tr>
</tbody>
</table>
An assignment $\alpha$ is autark for $F$ if $F\alpha \subseteq F$.

Generalizes pure literals:
- Assignment $[a := 1]$ is autark for $F$ iff $a$ is pure in $F$.

**Property**

If $\alpha$ is autark for $F$,
then $F$ is satisfiable iff there is $\beta \supseteq \alpha$ with $\beta \models F$. 
Monien-Speckenmeyer algorithm

\[ MS(F, \alpha) \]

1. if \( F\alpha = 0 \) then return UNSAT
2. if \( F\alpha = 1 \) then return \( \alpha \)

pick shortest clause \( C = a_1 \lor \ldots \lor a_r \)

for \( i := 1 \) to \( r \) do

3. \( \gamma_i := [a_1 := 0, \ldots, a_{i-1} := 0, a_i := 1] \)
4. if \( \gamma_i \) autark for \( F\alpha \) then return \( MS(F, \alpha \cup \gamma_i) \)

for \( i := 1 \) to \( r \) do

5. \( \beta := MS(F, \alpha \cup \gamma_i) \)
6. if \( \beta \neq UNSAT \) then return \( \beta \)

return UNSAT
Analysis of Monien-Speckenmeyer

\[ T(n): \text{ tree-size for formulas in } n \text{ variables} \]

\[ T'(n): \text{ tree-size for formulas in } n \text{ variables with a clause } C \text{ of width } w(C) \leq k - 1 \]

\[ T(n) = \max(T(n - 1), T'(n - 1) + \ldots + T'(n - k)) \]

\[ T'(n) = \max(T(n - 1), T'(n - 1) + \ldots + T'(n - k + 1)) \]

**Lemma**

\[ T(n) \leq T'(n) + T'(n - 1) \text{ for every } n. \]

Setting \( T'(n) = b^n \) yields: \( b^{k-1} = b^{k-2} + \ldots + b + 1 \)

and \( T(n) = O(b^n) \) for \( b = b_k \):

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_k )</td>
<td>1.61804</td>
<td>1.83929</td>
<td>1.92757</td>
<td>1.96595</td>
<td>1.98359</td>
<td>1.99197</td>
</tr>
</tbody>
</table>
Branching tuples

For a vector \((d_1, \ldots, d_m)\) with \(m \geq 1\) and \(d_i > 0\) let \(\tau(d_1, \ldots, d_m)\) be the unique positive solution to:

\[ x^{-d_1} + \ldots + x^{-d_m} = 1 \]

For a tree \(T\) with edge valuation \(d\):

- for a vertex \(v\) with children \(w_1, \ldots, w_m\):
  \[ d(v) = (d_1, \ldots, d_m) \text{ where } d_i = d(v, w_i). \]

- branching number \(\tau(v) := \tau(d(v))\)

- \(\tau(T) := \max_v(\tau(v))\) over the inner vertices \(v\)

- for a path \(p = v_1, \ldots, v_\ell\) let \(d(p) := \sum_{i=1}^{\ell-1} d(v_i, v_{i+1})\)

- \(d(T) := \max_p d(p)\) over all paths from the root to a leaf
Analysing DPLL with branching tuples

**Theorem**

*The number of leaves in \( T \) is at most \( \tau(T)^{d(T)} \).*

To analyse a DPLL algorithm:

- find distance \( d \) function for the recursion tree s.t.
  - \( d(T) \leq n \)
  - \( \tau(T) \) is minimized
To analyse simpleMS for 3-SAT, define:

\[ d(v, w) := \text{number of variables set}. \]

\[ d(p) \leq n \text{ for every path.} \]

Inner vertex \( v \) has 3 children of distance 3, 2, 1 resp.

Thus: \( \tau(v) = \tau(3, 2, 1) \) is the solution of

\[ x^{-3} + x^{-2} + x^{-1} = 1 \]

Multiplying by \( x^3 \) yields \( x^3 = x^2 + x + 1 \) as before

Thus \( \tau(T) = \tau(v) = 1.83929 \ldots \)
There are three types of inner vertices:

- **Type 1: autark assignment:**
  \( \nu \) has one child of distance 1:
  \[ \tau(\nu) = \tau(1) = 1 \]

- **Type 2: after non-autark assignment:**
  \( \nu \) has 2 children of distance 2 and 1:
  \[ \tau(\nu) = \tau(2, 1) = 1.61804 \ldots \]

- **Type 3: after autark assignment:**
  \( \nu \) has 3 children of distance 3, 2 and 1:
  \[ \tau(\nu) = \tau(3, 2, 1) = 1.83929 \ldots \]

\[ \Rightarrow \text{no improvement over simpleMS.} \]
Analysing Monien-Speckenmeyer

Vertex \( v \) of type 3 only as child of \( v' \) of type 1.

Idea: merge \( v \) and \( v' \) together to one vertex \( w \)
\( w \) has 3 children of distance 4, 3 and 2:
\[
\tau(w) = \tau(4, 3, 2) = 1.46558\ldots
\]
Thus \( \tau(T) = \tau(2, 1) = 1.61804\ldots \)

Alternative preserving tree structure: redefine distances
- for type 1: \( d(v, v') = 1 - \epsilon \)
- for type 3: \( d(v, w_i) = i + \epsilon \) for \( i = 1, 2, 3 \)

\( d(p) \) remains unchanged for every path \( p \).

Now for \( \epsilon = 0.5 \) we have for \( v \) of type 3:
\[
\tau(v) = \tau(3.5, 2.5, 1.5) = 1.59074\ldots < \tau(2, 1).
\]
Properties of branching numbers

- If $e > d_1$, then $\tau(e, d_2, \ldots, d_m) < \tau(d_1, \ldots d_m)$.

- If $d_1 + d_2 = e_1 + e_2$, and $\min(d_1, d_2) \geq \min(e_1, e_2)$, then
  \[
  \tau(d_1, \ldots, d_m) \leq \tau(e_1, e_2, d_3, \ldots, d_m)
  \]
  where equality only holds if it holds in the premise.

- Let $d = (d_1, \ldots, d_m)$ and $e := (e_1, \ldots, e_n)$
  and define $d^* := (d_1 + e_1, \ldots, d_1 + e_n, d_2, \ldots, d_m)$.
  Then:
    - if $\tau(d) \leq \tau(e)$, then $\tau(d) \leq \tau(d^*) \leq \tau(e)$.
    - if $\tau(d) \geq \tau(e)$, then $\tau(d) \geq \tau(d^*) \geq \tau(e)$.

Both inequalities are strict if those in the premise are.
Idea for algorithm of Zhang

Idea: Guarantee the existence of short clauses!

Thus: no uncontrolled unit propagation or pure literal elimination.

- $U$: the set of unit clauses,
  $$u := |U|$$

- $D$: a maximal set of 2-clauses variable-disjoint to $U$
  and among themselves
  $$d := |D|$$

- $T$: the remaining 2-clauses
  $$t := \min(|T|, 2)$$
The algorithm uses the following simplification rules:

- if there is $x$ with $x \in U$ and $\bar{x} \in U$ return UNSAT.
- delete subsumed 3-clauses.
- if $u \geq 2$ or $u = 1$ and $d > 0$,
  pick unit clause $a$ and set $[a := 1]$

An assignment $\alpha$ is quasi-autark for $F$,
if $|\text{dom } \alpha| = 1$ and $|F \alpha \setminus F| = 1$. 
Algorithm of Zhang

\( \text{Zh}(F, \alpha) \)

simplify\((F, \alpha)\)
if \( F = 0 \) then return UNSAT
if \( F = 1 \) then return \( \alpha \)
if \( u = 1 \) then return unit\((F, \alpha)\)
\((\gamma_1, \gamma_2) := \text{branch}(F, \alpha)\)
if \( \gamma_i \) autark for \( F \)
then return aut\((F, \gamma_i, \alpha)\)
if \( \gamma_i \) quasi-autark for \( F \)
then return qu-aut\((F, \gamma_i, \alpha)\)
\( \beta := \text{Zh}(FG_1, \alpha \cup \gamma_1) \)
if \( \beta \neq \text{UNSAT} \)
then return \( \beta \)
else return \( \text{Zh}(FG_2, \alpha \cup \gamma_2) \)
Autarkies and the last unit

After simplification we have \( u = 0 \), or \( u = 1 \) and \( d = 0 \).

\[
\text{unit}(F, \alpha) = \begin{cases} 
\text{let } a \in U \\
\text{pick a 3-clause } (b \lor c \lor d) \\
\gamma := [a := 1, b := 0, c := 0, d := 1] \\
\text{if } \gamma \text{ autark for } F \\
\quad \text{then return } \text{aut}(F, \gamma, \alpha) \\
\beta := \text{Zh}(F[a := 1] \land (b \lor c), \alpha \cup [a := 1]) \\
\text{if } \beta \neq \text{UNSAT} \\
\quad \text{then return } \beta \\
\quad \text{else return } \text{Zh}(F \gamma, \alpha \cup \gamma) 
\end{cases}
\]

\[
\text{aut}(F, \gamma, \alpha) = \begin{cases} 
\text{let } \gamma = \gamma' \cup [a \leftarrow 1] \\
\text{return } \text{Zh}(F \gamma' \land a, \alpha \cup \gamma') 
\end{cases}
\]
The branching

Case 1: $t \geq 1$, there are 2-clauses with common variables

Case 1.1: there are 2-clauses $(a \lor b)$ and $(\overline{a} \lor c)$

$\gamma_1 := [a := 1, c := 1]$
$\gamma_2 := [a := 0, b := 1]$

Case 1.2: otherwise pick 2-clauses $(a \lor b)$ und $(a \lor c)$

$\gamma_1 := [a := 1]$
$\gamma_2 := [a := 0, b := 1, c := 1]$

Case 2: $t = 0$, all 2-clauses are variable-disjoint, pick one $(a \lor b)$

$\gamma_1 := [a := 1]$
$\gamma_2 := [a := 0, b := 1]$
Some lemmas

\[ F\langle a := C \rangle \] denotes \( F \) with \( a \) replaced by \( C \) everywhere.

**Lemma**

*If \( a \) does not occur in \( F \), then \( F \land (a \lor C) \) is satisfiable iff \( F\langle \bar{a} := C \rangle \) is.*

**Corollary**

*If \( a \) occurs in \( F \) only in \((a \lor b \lor C)\), then \( F \) is satisfiable iff one of the following is:*

\[ F[b := 1, a := 0] \quad F\langle \bar{a} := C \rangle[b := 0, a := 1] \]

**Lemma**

*If \( a \) und \( b \) do not occur in \( F \), then \( F \land (a \lor b \lor c) \) is satisfiable iff one of the following is:*

\[ F[c \leftarrow 1, a \leftarrow 0, b \leftarrow 0] \]
\[ F[c \leftarrow 0, a \leftarrow 1, b \leftarrow 0] \]
\[ F[c \leftarrow 0, a \leftarrow 0, b \leftarrow 1] \]
Treating quasi-autarkies

qu-aut($F, \gamma, \alpha$)

let $\gamma = [a := 1]$ and $F \gamma \setminus F = \{(b \lor c)\}$

if $[b := 1, a := 1]$ autark for $F$

then return $\text{aut}(F, [b := 1, a := 1], \alpha)$

if $F\langle a := c\rangle[b := 0, a := 0] \subseteq F$

then $\gamma_1 := [c := 1, a := 1, b := 0]$

$\gamma_2 := [c := 0, a := 0, b := 0]$

$\gamma_3 := [c := 0, a := 1, b := 1]$

if there is 2-clause $(\bar{b} \lor d)$ with $d \notin \{a, \bar{a}, c, \bar{c}\}$

then $\gamma_3 := \gamma_3 \cup [d := 1]$

if $\gamma_i$ autark for $F$

then return $\text{aut}(F, \gamma_i, \alpha)$

for $i := 1$ to 3

$\beta := \text{Zh}(F\gamma_i, \alpha \cup \gamma_i)$

if $\beta \neq \text{UNSAT}$ then return $\beta$

return UNSAT

$\beta := \text{Zh}(F[b := 1, a := 1], \alpha \cup [b := 1, a := 1])$

if $\beta \neq \text{UNSAT}$

then return $\beta$

else return $\text{Zh}(F\langle a := c\rangle[b := 0, a := 0], \alpha \cup [b := 0, a := 0])$
The distance function

For a formula $F$ we define the measure $\mu$

- $\mu = n - \epsilon(u + d + t)$

For nodes $v$ and $v'$ with formulas of measure $\mu$ and $\mu'$

- $d(v, v') = \mu - \mu'$
Analysis of \texttt{unit}()

Invoked only when $u = 1$, $d = 0$.

First branch: $d(v, v') = 1$
- one variable assigned: $n \rightsquigarrow n - 1$
- unit clause satisfied: $u \rightsquigarrow 0$
- all 2-clauses become units: $u \rightsquigarrow t$, $t \rightsquigarrow 0$
- one 2-clause added: $d \rightsquigarrow 1$

Second branch $d(v, v') = 4 - 2\epsilon$
- four variables assigned: $n \rightsquigarrow n - 4$
- unit clause satisfied: $u \rightsquigarrow 0$
- all 2-clauses satisfied: $t \rightsquigarrow 0$
- no autarky, thus one 2-clause added: $d \rightsquigarrow 1$
Analysis of the main branches

Case 1.1: \( \tau(v) = (2 - 3\epsilon, 2 - 3\epsilon) \)

In both branches: \( d(v, v') = 2 - 3\epsilon \)
- two variables assigned: \( n \leadsto n - 2 \)
- two 2-clauses in \( D \) satisfied: \( d \leadsto d - 2 \)
- all 2-clauses in \( T \) satisfied: \( t \leadsto 0 \)
- no autarky, thus one 2-clause added: \( d \leadsto d + 1 \)
Analysis of the main branches

Case 1.2: $\tau(v) = (1 - \epsilon, 3 - 3\epsilon)$

First branch: $d(v, v') = 1 - \epsilon$
  ▶ one variable assigned: $n \sim n - 1$
  ▶ one 2-clause from $D$ satisfied: $d \sim d - 1$
  ▶ all 2-clauses from $T$ satisfied: $t \sim t - 2$
  ▶ not quasi-autark, thus 2 new 2-clauses

Second branch $d(v, v') = 3 - 3\epsilon$
  ▶ three variables assigned: $n \sim n - 3$
  ▶ two 2-clause from $D$ satisfied: $d \sim d - 1$
  ▶ all 2-clauses from $T$ satisfied: $t \sim t - 2$
  ▶ no autarky, thus one new 2-clause
Analysis of the main branches

Case 2: $\tau(v) = (1 + \epsilon, 2)$

First branch: $d(v, v') = 1 + \epsilon$
  - one variable assigned: $n \rightsquigarrow n - 1$
  - one 2-clause from $D$ satisfied: $d \rightsquigarrow d - 1$
  - not quasi-autark, thus 2 new 2-clauses

Second branch: $d(v, v') = 2$
  - two variables assigned: $n \rightsquigarrow n - 2$
  - one 2-clause from $D$ satisfied: $d \rightsquigarrow d - 1$
  - no autarky, thus one new 2-clause
Analysis of \texttt{qu-aut()}

Case (A): \( \tau(v) = (3 - 3\epsilon, 3 - 3\epsilon, 3 - 3\epsilon) \)

- in every branch: \( n \leadsto n - 3, \quad d \leadsto d - 2, \quad t \leadsto t - 2 \)
- no autarky, thus always one new 2-clause.

Case (B): \( \tau(v) = (3 - 4\epsilon, 3 - 4\epsilon, 4 - 4\epsilon) \)

- in every branch: \( n \leadsto n - 3, \quad d \leadsto d - 3, \quad t \leadsto t - 2 \)
- no autarky, thus always one new 2-clause.
- in the final branch, one additional variable assigned.

Case (C):
- analogous to Case 1.1
The final analysis

The following branching tuples occur:

\[ \tau(1, 4 - 2\epsilon) \]  Procedure unit()
\[ \tau(2 - 3\epsilon, 2 - 3\epsilon) \]  Case 1.1, procedure qu-aut(), case (C)
\[ \tau(1 - \epsilon, 3 - 3\epsilon) \]  Case 1.2
\[ \tau(1 + \epsilon, 2) \]  Case 2
\[ \tau(3 - 3\epsilon, 3 - 3\epsilon, 3 - 3\epsilon) \]  Procedure qu-aut(), case (A)
\[ \tau(3 - 4\epsilon, 3 - 4\epsilon, 4 - 4\epsilon) \]  Procedure qu-aut(), case (B)

The maximum is smallest for \( \epsilon = 0.1528477 \), where
\[ \tau(T) = \tau(1 + \epsilon, 2) < 1.570214. \]
Upper bounds for 3-SAT

Theorem

Zhang’s algorithm solves 3-SAT in time $O(1.57022^n)$

Better bounds for 3-SAT were obtained with similar, more complex algorithms and analyses:

- Kullmann (1993): $O(1.5045^n)$
- Schiermeyer (1996): $O(1.4963^n)$