Overview

The basic DPLL algorithm

After *Davis, Putnam, Logemann* and *Loveland*, 1960

The basic DPLL Algorithm:

DPLL(F,α)

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

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

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

if $\beta \neq$ UNSAT
then return β
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 $b_k := \sqrt[k]{2^k - 1} < 2$

k	3	4	5	6	7	8
b_k	1.91294	1.96799	1.98735	1.99477	1.99777	1.99903

Unit propagation

A branching strategy (almost) always employed:

- pick x if x occurs in a unit clause a.

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

Mostly realized as simplification step before branching:

```
UnitProp(F,α)
  while Fα contains unit clause a
     α := α ∪ [a := 1]
  return α
```

Pure literals and subsumption

Other simplification steps in original DPLL:

- elimination of pure literals
- deletion of subsumed clauses

`PureLit(`F`,`α`)`

`while` $F\alpha$ `contains pure literal` a

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

Clause C subsumes D if $C \subseteq D$

`Subs(`F`)`

`while` F `contains clauses` $C \subseteq D$

$F := F \setminus D$

The general DPLL algorithm

```
DPLL(F, α)
  simplify(F, α)
  if F = 0 then return UNSAT
  if F = 1 then return α
  pick x ∈ V(F) and ε ∈ {0,1}
  β := DPLL(F[x := ε], α ∪ [x := ε])
  if β ≠ UNSAT
     then return β
     else return DPLL(F[x := ε̄], α ∪ [x := ε̄])
```

Monien-Speckenmeyer algorithm - simple version

Branching strategy:

- pick literal from a shortest clause

Equivalently:

simpleMS(F,α)

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

pick shortest clause $C = a_1 \vee \ldots \vee a_r$

for $i := 1$ to r do

$\gamma_i := [a_1 := 0, \ldots, a_{i-1} := 0, a_i := 1]$
$\beta :=$ simpleMS($F, \alpha \cup \gamma_i$)
if $\beta \neq$ UNSAT then return β

return UNSAT

Analysis of `simpleMS`

At each node: r-fold branching for some $r \leq k$,

in ith 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:

k	3	4	5	6	7	8
b_k	1.83929	1.92757	1.96595	1.98359	1.99197	1.99603

Autarkies

An assignment α 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 α is autark for F,
then F is satisfiable iff there is $\beta \supseteq \alpha$ with $\beta \models F$.

Monien-Speckenmeyer algorithm

MS(F,α)

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

pick shortest clause $C = a_1 \vee \ldots \vee a_r$

for $i := 1$ to r do

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

for $i := 1$ to r do

$\beta :=$ MS($F, \alpha \cup \gamma_i$)
if $\beta \neq$ UNSAT then return β

return UNSAT

Analysis of Monien-Speckenmeyer

$T(n)$: tree-size for formulas in n variables

$T'(n)$: tree-size for formulas in n variables
with a clause C 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

$T(n) \leq T'(n) + T'(n-1)$ 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 $b = b_k$:

k	3	4	5	6	7	8
b_k	1.61804	1.83929	1.92757	1.96595	1.98359	1.99197

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 $w_1, \ldots, w_m$:
  $d(v) = (d_1, \ldots, d_m)$ 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

`simpleMS` revisited

To analyse `simpleMS` for 3-SAT, define:

$d(v, w) :=$ number of variables set.

$d(p) \leq n$ 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 $\tau(T) = \tau(v) = 1.83929\ldots$

Analysing Monien-Speckenmeyer

There are three types of inner vertices:

- Type 1: autark assignment:
  v has one child of distance 1:
  $$\tau(v) = \tau(1) = 1$$
- Type 2: after non-autark assignment:
  v has 2 children of distance 2 and 1:
  $$\tau(v) = \tau(2, 1) = 1.61804\ldots$$
- Type 3: after autark assignment:
  v has 3 children of distance 3, 2 and 1:
  $$\tau(v) = \tau(3, 2, 1) = 1.83929\ldots$$

$\rightsquigarrow$ 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
  $t := min(|T|, 2)$

Simplifications employed

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 α is quasi-autark for F,
if $|\operatorname{dom} \alpha| = 1$ and $|F\alpha \setminus F| = 1$.

Algorithm of Zhang

$\mathrm{Zh}(F, \alpha)$

```
simplify(F, α)
if F = 0 then return UNSAT
if F = 1 then return α
if u = 1 then return unit(F, α)
(γ1, γ2) := branch(F, α)
if γi autark for F
    then return aut(F, γi, α)
if γi quasi-autark for F
    then return qu-aut(F, γi, α)
β := Zh(Fγ1, α ∪ γ1)
if β ≠ UNSAT
    then return β
    else return Zh(Fγ2, α ∪ γ2)
```

Autarkies and the last unit

After simplification we have $u = 0$, or $u = 1$ and $d = 0$.

$\texttt{unit}(F, \alpha)$

  $\texttt{let}\ a \in U$

  $\texttt{pick a 3-clause}\ (b \vee c \vee d)$

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

  $\texttt{if}\ \gamma\ \texttt{autark for}\ F$

    $\texttt{then return aut}(F, \gamma, \alpha)$

  $\beta := \mathrm{Zh}(F[a := 1] \wedge (b \vee c), \alpha \cup [a := 1])$

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

    $\texttt{then return}\ \beta$

    $\texttt{else return Zh}(F\gamma, \alpha \cup \gamma)$

$\texttt{aut}(F, \gamma, \alpha)$

  $\texttt{let}\ \gamma = \gamma' \cup [a \leftarrow 1]$

  $\texttt{return}\ Zh(F\gamma' \wedge a, \alpha \cup \gamma')$

The branching

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

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

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

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

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

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

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

Some lemmas

$F\langle a := C\rangle$ denotes F with a replaced by C everywhere.

Lemma

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

Corollary

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

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

Lemma

If a und b do not occur in F, then $F \wedge (a \vee b \vee c)$ 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, γ, α)
  let γ = [a := 1] and Fγ \ F = {(b ∨ c)}
  if [b := 1, a := 1] autark for F
      then return aut(F, [b := 1, a := 1], α)
  if F⟨a := c⟩[b := 0, a := 0] ⊆ F
      then γ1 := [c := 1, a := 1, b := 0]
           γ2 := [c := 0, a := 0, b := 0]                    (A)
           γ3 := [c := 0, a := 1, b := 1]
           if there is 2-clause (b̄ ∨ d) with d ∉ {a, ā, c, c̄}
               then γ3 := γ3 ∪ [d := 1]                      (B)
           if γi autark for F
               then return aut(F, γi, α)
           for i := 1 to 3
               β := Zh(Fγi, α ∪ γi)
               if β ≠ UNSAT then return β
           return UNSAT
  β := Zh(F[b := 1, a := 1], α ∪ [b := 1, a := 1])          (C)
  if β ≠ UNSAT
      then return β
      else return Zh(F⟨a := c⟩[b := 0, a := 0], α ∪ [b := 0, a := 0])
```

The distance function

For a formula F we define the measure μ

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

For nodes v and v' with formulas of measure μ and μ'

- $d(v, v') = \mu - \mu'$

Analysis of `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: $n \rightsquigarrow n - 2$
- two 2-clauses in D satisfied: $d \rightsquigarrow d - 2$
- all 2-clauses in T satisfied: $t \rightsquigarrow 0$
- no autarky, thus one 2-clause added: $d \rightsquigarrow d + 1$

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: $n \rightsquigarrow n-1$
- one 2-clause from D satisfied: $d \rightsquigarrow d-1$
- all 2-clauses from T satisfied: $t \rightsquigarrow t-2$
- not quasi-autark, thus 2 new 2-clauses

Second branch $d(v, v') = 3 - 3\epsilon$

- three variables assigned: $n \rightsquigarrow n-3$
- two 2-clause from D satisfied: $d \rightsquigarrow d-1$
- all 2-clauses from T satisfied: $t \rightsquigarrow t-2$
- 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 `qu-aut()`

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

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

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

- in every branch: $n \rightsquigarrow n - 3, \quad d \rightsquigarrow d - 3, \quad t \rightsquigarrow t - 2$
- 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): $O(1.4963^n)$