



**Off the trail:
re-examining the CDCL algorithm.**

**Alexandra Goultiaeva and Fahiem Bacchus
University of Toronto**

SAT 2012, Trento, Italy

Overview

- Using the standard (historical) description, the trail seems vital to the completeness of DPLL.
- Paradigms and descriptions change
- The trail is not as important as it may seem
 - Is simply an efficient way of deciding which variable to flip
 - No inherent importance to the algorithm
- Current CDCL algorithms can be reformulated as local search with a complicated heuristic

DPLL Algorithm

Given: ϕ - a formula in CNF

Solve (ϕ)

$\phi \leftarrow \text{unitPropagate}(\phi)$

if ϕ reduces to a constant, return ϕ

$v \leftarrow$ 'best' unassigned var

return $\text{Solve}(\phi|_v) \vee \text{Solve}(\phi|_{\neg v})$

DPLL Algorithm

Given: ϕ - a formula in CNF

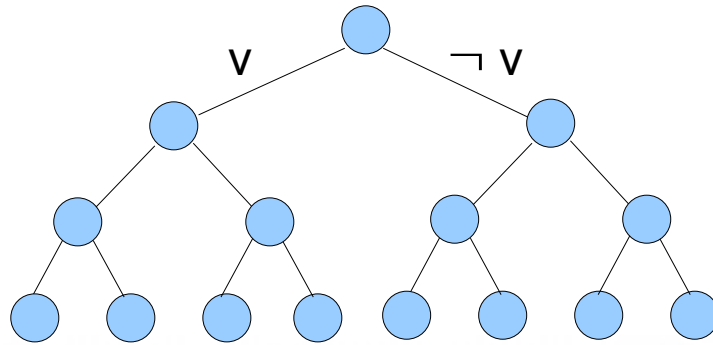
Solve (ϕ)

$\phi \leftarrow \text{unitPropagate}(\phi)$

if ϕ reduces to a constant, return ϕ

$v \leftarrow$ 'best' unassigned var

return $\text{Solve}(\phi|_v) \vee \text{Solve}(\phi|_{\neg v})$



DPLL Algorithm

Given: ϕ - a formula in CNF

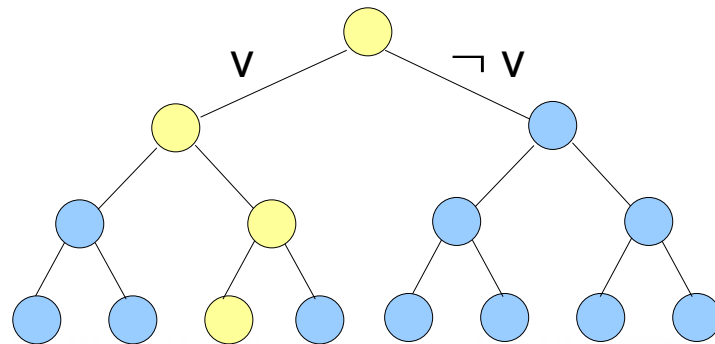
Solve (ϕ)

$\phi \leftarrow \text{unitPropagate}(\phi)$

if ϕ reduces to a constant, return ϕ

$v \leftarrow$ 'best' unassigned var

return $\text{Solve}(\phi|_v) \vee \text{Solve}(\phi|_{\neg v})$



DPLL Algorithm

Given: ϕ - a formula in CNF
 $\pi \leftarrow \emptyset$

while (TRUE)

UnitPropagate(ϕ , π)

if reduce(ϕ , π) contains an empty clause

if no decisions **then** return FALSE

$v \leftarrow$ last decision

$\pi \leftarrow$ backtrack(lvl[v])

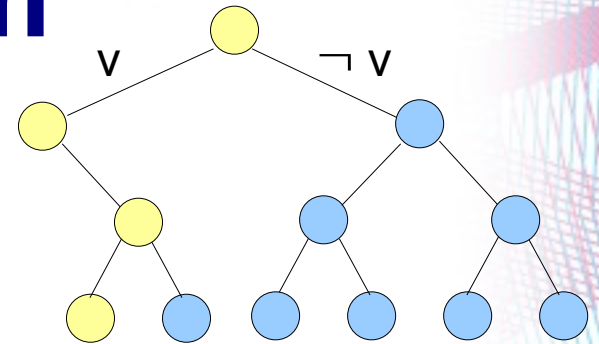
π .append($\neg v$)

else if ϕ is TRUE under π **then** return TRUE

else

$v \leftarrow$ 'best' unassigned var

π .append(v)



DPLL Algorithm

Given: ϕ - a formula in CNF
 $\pi \leftarrow \emptyset$

while (TRUE)

UnitPropagate(ϕ , π)

if reduce(ϕ , π) contains an empty clause

if no decisions **then** return FALSE

$v \leftarrow$ last decision

$\pi \leftarrow$ backtrack(lvl[v])

π .append($\neg v$)

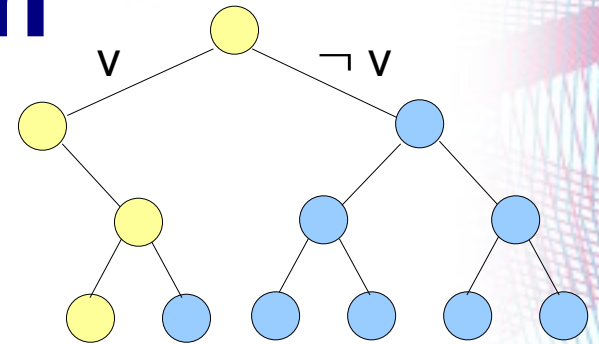
else if ϕ is TRUE under π **then** return TRUE

else

$v \leftarrow$ 'best' unassigned var

π .append(v)

if timeToRestart() **then** backtrack(0)



DPLL Algorithm

Given: ϕ - a formula in CNF
 $\pi \leftarrow \emptyset$

while (TRUE)

UnitPropagate(ϕ , π)

if reduce(ϕ , π) contains an empty clause

if no decisions **then** return FALSE

$v \leftarrow$ last decision

$\pi \leftarrow$ backtrack(lvl[v])

π .append($\neg v$)

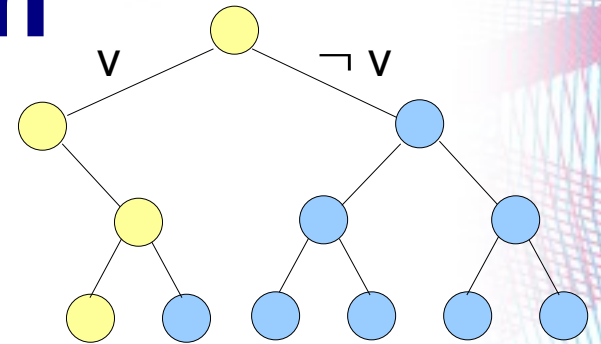
else if ϕ is TRUE under π **then** return TRUE

else

$v \leftarrow$ 'best' unassigned var

π .append(v)

if timeToRestart() **then** backtrack(0)



DPLL Algorithm

Given: ϕ - a formula in CNF

$\pi \leftarrow \emptyset$

$C \leftarrow \emptyset$

while (TRUE)

UnitPropagate($\phi \cup C$, π)

if reduce($\phi \cup C$, π) contains an empty clause

$c' \leftarrow$ clauseLearn($\phi \cup C$, π)

if $c' = \emptyset$ **then** return FALSE

$C \leftarrow C \cup \{c'\}$

$\pi \leftarrow$ backtrack(c')

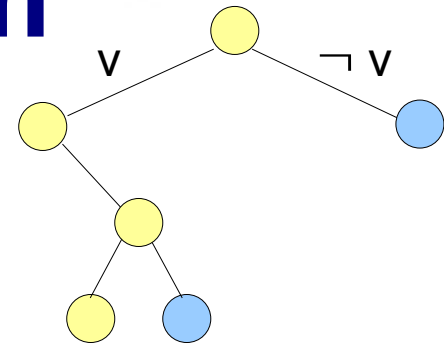
else if ϕ is TRUE under π **then** return TRUE

else

$v \leftarrow$ 'best' unassigned var

π .append(v)

if timeToRestart() **then** backtrack(0)



DPLL Algorithm

Given: ϕ - a formula in CNF

$\pi \leftarrow \emptyset$

$C \leftarrow \emptyset$

while (TRUE)

UnitPropagate($\phi \cup C, \pi$)

if reduce($\phi \cup C, \pi$) contains an empty clause

$c' \leftarrow \text{clauseLearn}(\phi \cup C, \pi)$

if $c' = \emptyset$ **then** return FALSE

$C \leftarrow C \cup \{c'\}$

$\pi \leftarrow \text{backtrack}(c')$

else if ϕ is TRUE under π **then** return TRUE

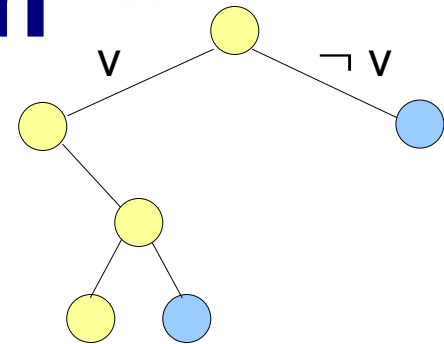
else **Clause learning:**

- Learn from conflicts

- *Empowering* clauses

- Without deletion,

- guarantees completeness



DPLL Algorithm

Given: ϕ - a formula in CNF

$\pi \leftarrow \emptyset$

$C \leftarrow \emptyset$

while (TRUE)

UnitPropagate($\phi \cup C$, π)

if reduce($\phi \cup C$, π) contains an empty clause

$c' \leftarrow$ clauseLearn($\phi \cup C$, π)

if $c' = \emptyset$ **then** return FALSE

$C \leftarrow C \cup \{c'\}$

$\pi \leftarrow$ backtrack(c')

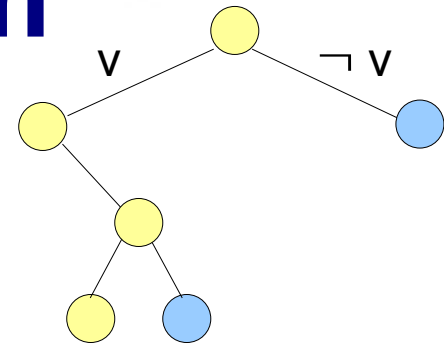
else if ϕ is TRUE under π **then** return TRUE

else

$v \leftarrow$ 'best' unassigned var

π .append(v)

if timeToRestart() **then** backtrack(0)



DPLL Algorithm

Given: ϕ - a formula in CNF

$\pi \leftarrow \emptyset$

$C \leftarrow \emptyset$

while (TRUE)

UnitPropagate($\phi \cup C, \pi$)

if reduce($\phi \cup C, \pi$) contains an empty clause

$c' \leftarrow$ clauseLearn($\phi \cup C, \pi$)

if $c' = \emptyset$ **then** return FALSE

$C \leftarrow C \cup \{c'\}$

$\pi \leftarrow$ backtrack(c')

else if ϕ is TRUE under π **then** return TRUE

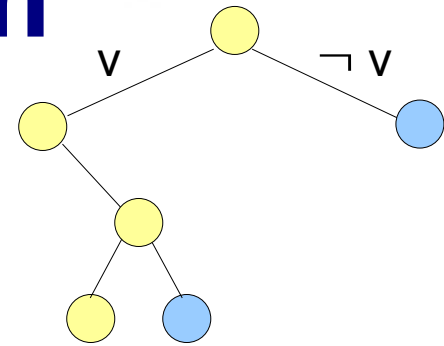
else

$v \leftarrow$ 'best' unassigned var

$v \leftarrow$ **phase**[v]

π .append(v)

if timeToRestart() **then** backtrack(0)



DPLL Algorithm

Given: ϕ - a formula in CNF

$\pi \leftarrow \emptyset$

$C \leftarrow \emptyset$

while (TRUE)

UnitPropagate($\phi \cup C, \pi$)

if reduce($\phi \cup C, \pi$) contains an empty clause

$c' \leftarrow$ clauseLearn($\phi \cup C, \pi$)

if $c' = \emptyset$ **then** return FALSE

$C \leftarrow C \cup \{c'\}$

$\pi \leftarrow$ backtrack(c')

else if ϕ is TRUE under π **then** return TRUE

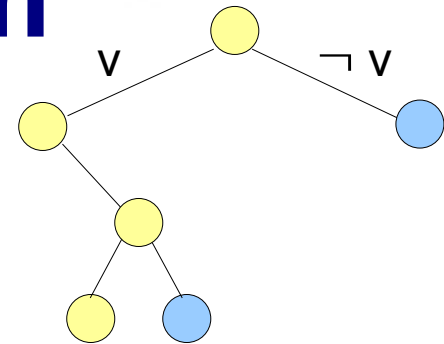
else

$v \leftarrow$ **unassigned var with largest VSIDS**

$v \leftarrow$ phase[v]

π .append(v)

if timeToRestart() **then** backtrack(0)



DPLL Algorithm

Given: ϕ - a formula in CNF

$\pi \leftarrow \emptyset$

$C \leftarrow \emptyset$

while (TRUE)

UnitPropagate($\phi \cup C, \pi$)

if reduce($\phi \cup C, \pi$) contains an empty clause

$c' \leftarrow$ clauseLearn($\phi \cup C, \pi$)

if $c' = \emptyset$ **then** return FALSE

$C \leftarrow C \cup \{c'\}$

$\pi \leftarrow$ backtrack(c')

else if ϕ is TRUE under π **then** return TRUE

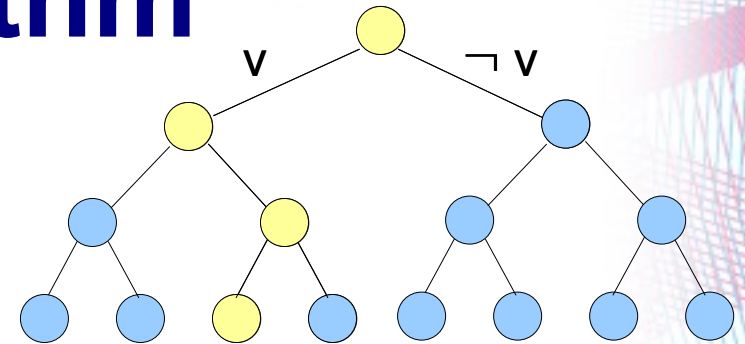
else

$v \leftarrow$ **unassigned var with largest VSIDS**

$v \leftarrow$ phase[v]

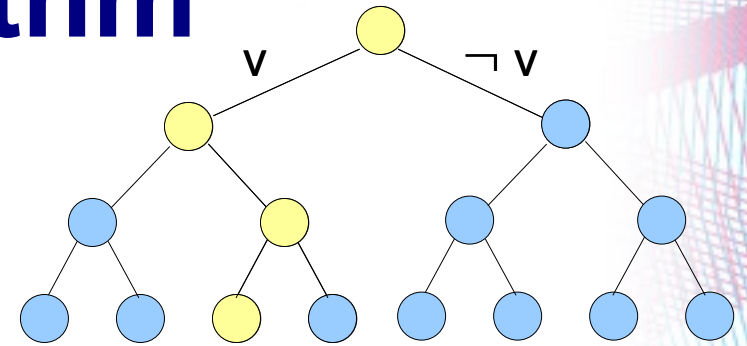
π .append(v)

if timeToRestart() **then** backtrack(0)



DPLL Algorithm

**Restart to a different part
of search space**



```
UnitPropagate( $\phi \cup C, \pi$ )  
if reduce( $\phi \cup C, \pi$ ) contains an empty clause  
   $c' \leftarrow$  clauseLearn( $\phi \cup C, \pi$ )  
  if  $c' = \emptyset$  then return FALSE  
   $C \leftarrow C \cup \{c'\}$   
   $\pi \leftarrow$  backtrack( $c'$ )
```

```
else if  $\phi$  is TRUE under  $\pi$  then return TRUE  
else
```

```
   $v \leftarrow$  unassigned var with largest VSIDS
```

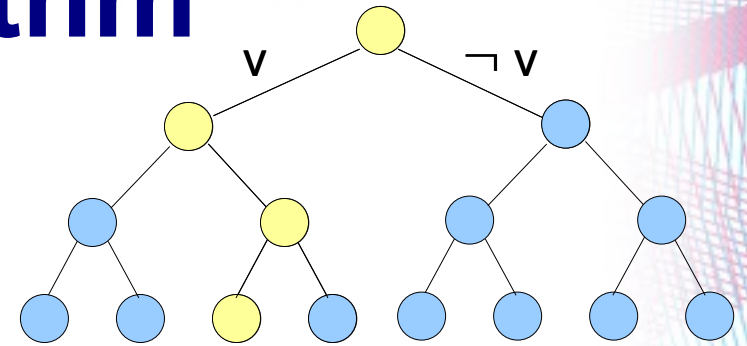
```
   $v \leftarrow$  phase[ $v$ ]
```

```
   $\pi$ .append( $v$ )
```

```
if timeToRestart() then backtrack(0)
```

DPLL Algorithm

**Restart to a different part
of search space
+ Progress saving**



```
if reduce( $\phi \cup C, \pi$ ) contains an empty clause  
   $c' \leftarrow \text{clauseLearn}(\phi \cup C, \pi)$   
  if  $c' = \emptyset$  then return FALSE  
   $C \leftarrow C \cup \{c'\}$   
   $\pi \leftarrow \text{backtrack}(c')$ 
```

```
else if  $\phi$  is TRUE under  $\pi$  then return TRUE  
else
```

```
   $v \leftarrow$  unassigned var with largest VSIDS
```

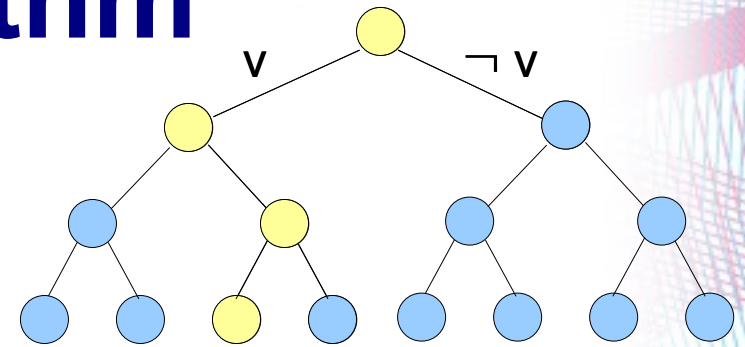
```
   $v \leftarrow \text{phase}[v]$ 
```

```
   $\pi.\text{append}(v)$ 
```

```
if timeToRestart() then backtrack(0)
```


DPLL Algorithm

**Restart to a different part
of search space
+ Progress saving
+ VSIDS**



```
contains an empty clause
c' ← clauseLearn( $\phi \cup C$ ,  $\pi$ )
if c' =  $\emptyset$  then return FALSE
C ← C  $\cup$  {c'}
 $\pi$  ← backtrack(c')
```

```
else if  $\phi$  is TRUE under  $\pi$  then return TRUE
else
```

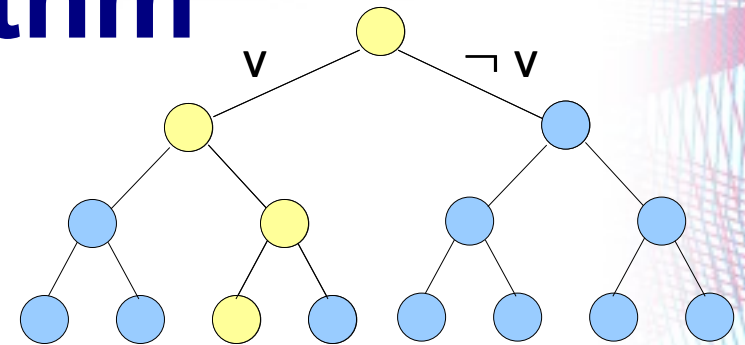
```
v ← unassigned var with largest VSIDS
```

```
v ← phase[v]
```

```
 $\pi$ .append(v)
```

```
if timeToRestart() then backtrack(0)
```

DPLL Algorithm



~~Restart to a different part
of search space~~

+ Progress saving
+ VSIDS

```
while  $\phi$  contains an empty clause  
do  
   $c' \leftarrow \text{clauseLearn}(\phi \cup C, \pi)$   
  if  $c' = \emptyset$  then return FALSE  
   $C \leftarrow C \cup \{c'\}$   
   $\pi \leftarrow \text{backtrack}(c')$ 
```

```
else if  $\phi$  is TRUE under  $\pi$  then return TRUE  
else
```

```
   $v \leftarrow$  unassigned var with largest VSIDS
```

```
   $v \leftarrow \text{phase}[v]$ 
```

```
   $\pi.\text{append}(v)$ 
```

```
if timeToRestart() then backtrack(0)
```

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

h (4.9)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

h (4.9)

i (2.3)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

h (4.9)

i (2.3)

e (3.2)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

h (4.9)

i (2.3)

e (3.2)

d (4.1)

f (2.3)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

h (4.9)

i (2.3)

e (3.2)

d (4.1)

f (2.3)

g (3.1)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

h (4.9)

i (2.3)

e (3.2)

d (4.1)

f (2.3)

g (3.1)

j (1.3)

k (2.8)

Re-creating the trail

a (5.4)

b (2.2)

c (2.5)

d (4.1)

e (3.2)

f (2.3)

g (3.1)

h (4.9)

i (2.3)

j (1.3)

k (2.8)

a (5.4)

b (2.2)

c (2.5)

h (4.9)

i (2.3)

e (3.2)

d (4.1)

f (2.3)

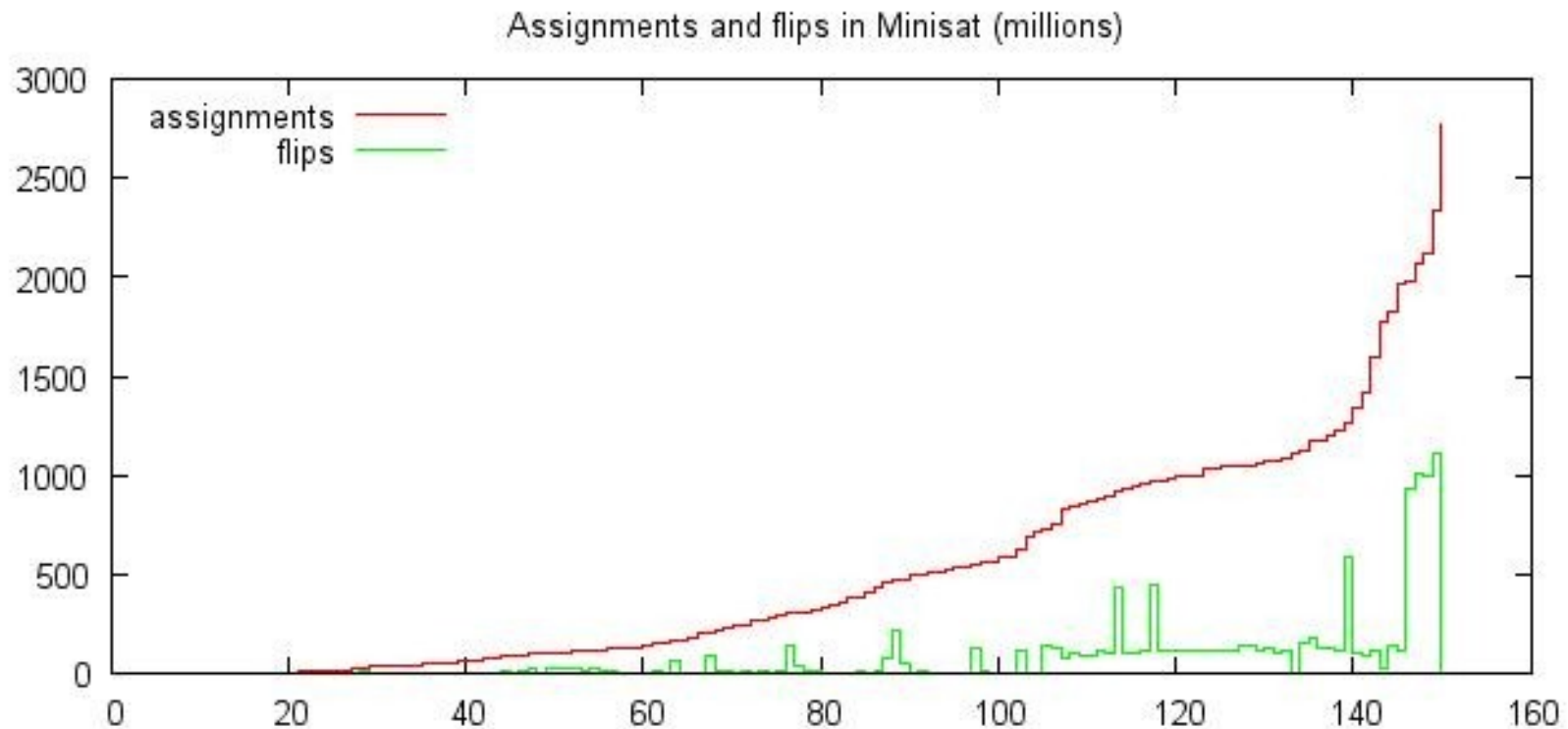
g (3.1)

j (1.3)

k (2.8)

Effect of restart: the trail is updated to
order decisions by heuristic

Assignments vs. flips



- **Assignment:** any time a variable gets a value
- **Flip:** any time a variable gets a value that is *different from its phase setting*

Concentrate on Flips

- Imagine a complete assignment:
 - Variable's value, if it exists
 - Otherwise, its phase setting

Concentrate on Flips

- Imagine a complete assignment:
 - Variable's value, if it exists
 - Otherwise, its phase setting
- Only flips change the complete assignment
 - Guaranteed a flip after every clause learning episode; or caused by other assignments

Concentrate on Flips

- Imagine a complete assignment:
 - Variable's value, if it exists
 - Otherwise, its phase setting
- Only flips change the complete assignment
 - Guaranteed a flip after every clause learning episode; or caused by other assignments
- Decision variables are never flipped
 - Every flip has a *reason*:

Concentrate on Flips

- Imagine a complete assignment:
 - Variable's value, if it exists
 - Otherwise, its phase setting
- Only flips change the complete assignment
 - Guaranteed a flip after every clause learning episode; or caused by other assignments
- Decision variables are never flipped
 - Every flip has a *reason*:
 - A clause which contains the flipped variable

Concentrate on Flips

- Imagine a complete assignment:
 - Variable's value, if it exists
 - Otherwise, its phase setting
- Only flips change the complete assignment
 - Guaranteed a flip after every clause learning episode; or caused by other assignments
- Decision variables are never flipped
 - Every flip has a *reason*:
 - A clause which contains the flipped variable
 - The flipped variable is the “least important” in that clause.

Intuition

- Variables with high heuristic values are “important”
- A false clause means a flip. We want to flip the least important variable.
- However: if $(\alpha \rightarrow x)$ and we flip x , then:
 - Either we'll need to flip x back
 - Or we'll need to flip a variable from α
- So: any variable implied by important variables is important

Local Search

Given: ϕ - a formula in CNF

```
while ( $\phi|_I$  is not TRUE)
```

```
    I  $\leftarrow$  initValues()
```

```
    while  $\phi|_I$  contains FALSE clauses
```

```
        if timeToRestart() then continue outer
```

```
        v  $\leftarrow$  pickVar(I)
```

```
        flip(v)
```

```
return TRUE
```

Local Search

- Want to get “closer to a solution”
 - estimating is very difficult
 - estimates are unreliable
 - lots of local minima
- Escaping local minima:
 - generate a new clause that would change the landscape
 - if a clause is guaranteed to be new, get completeness

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I ?
 - CDLS: “simulate UP until it first deviates from I ”

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

a
b
c
d
e
f
g
h

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

a
b
c
d
e
f
g
h

a

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

a
b
c
d
e
f
g
h

a
b
d
g

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

a
b
c
d
e
f
g
h

a
b
d
g
c

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

a
b
c
d
e
f
g
h

a
b
d
g
c
 $\neg e$

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

a
b
c
d
e
f
g
h

a
b
d
g
c
¬e

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

a
b
c
d
e
f
g
h

a
b
d
g
c
¬e

(e ¬d ¬f ¬h)

Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”

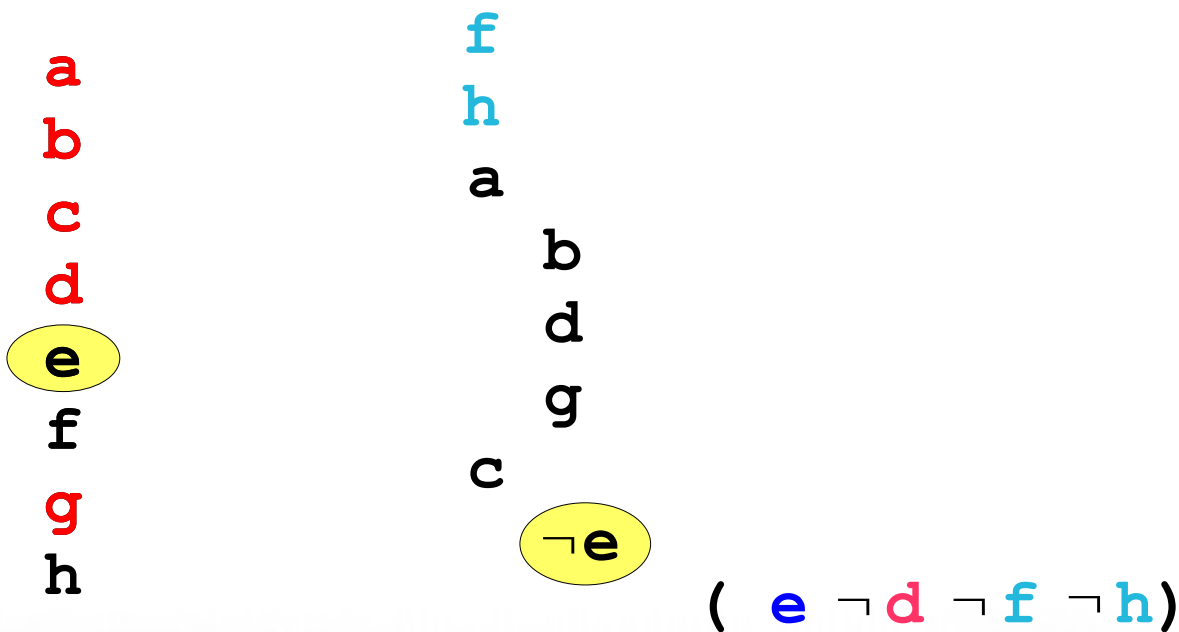
a
b
c
d
e
f
g
h

a
b
d
g
c
¬e

(e ¬ d ¬ f ¬ h)

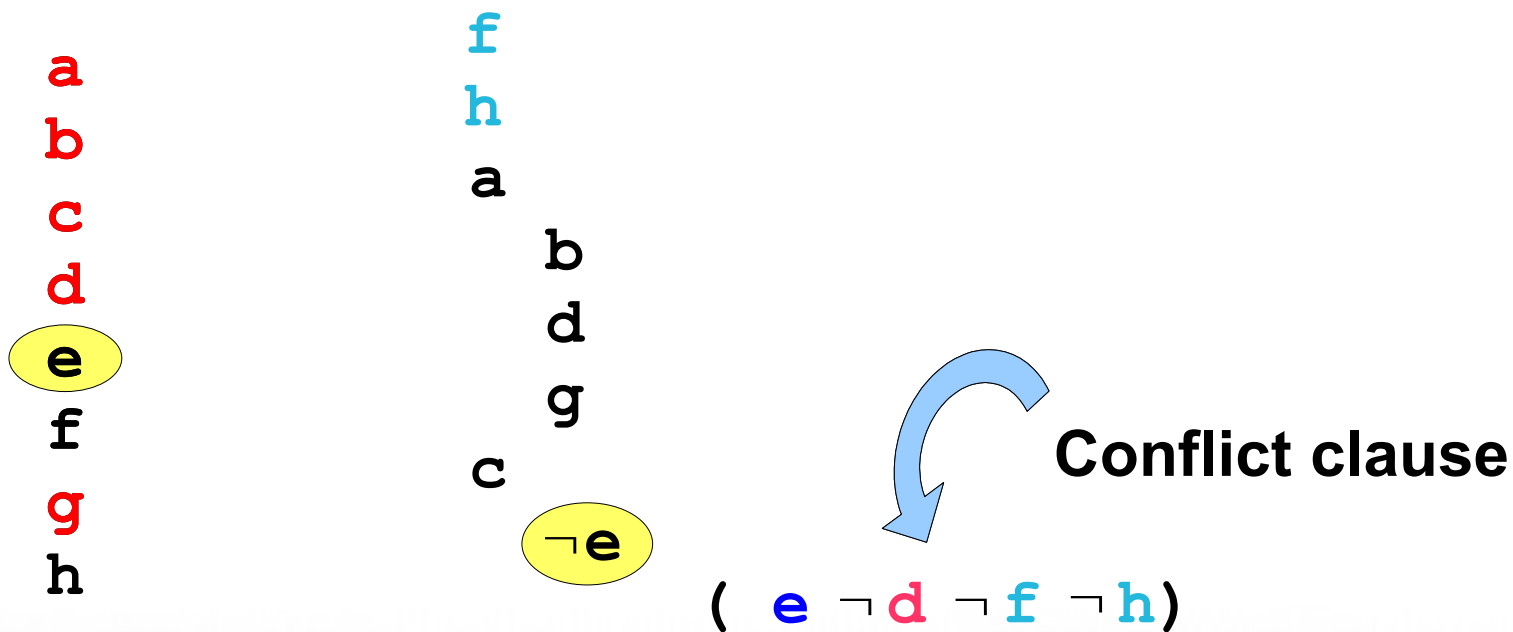
Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”



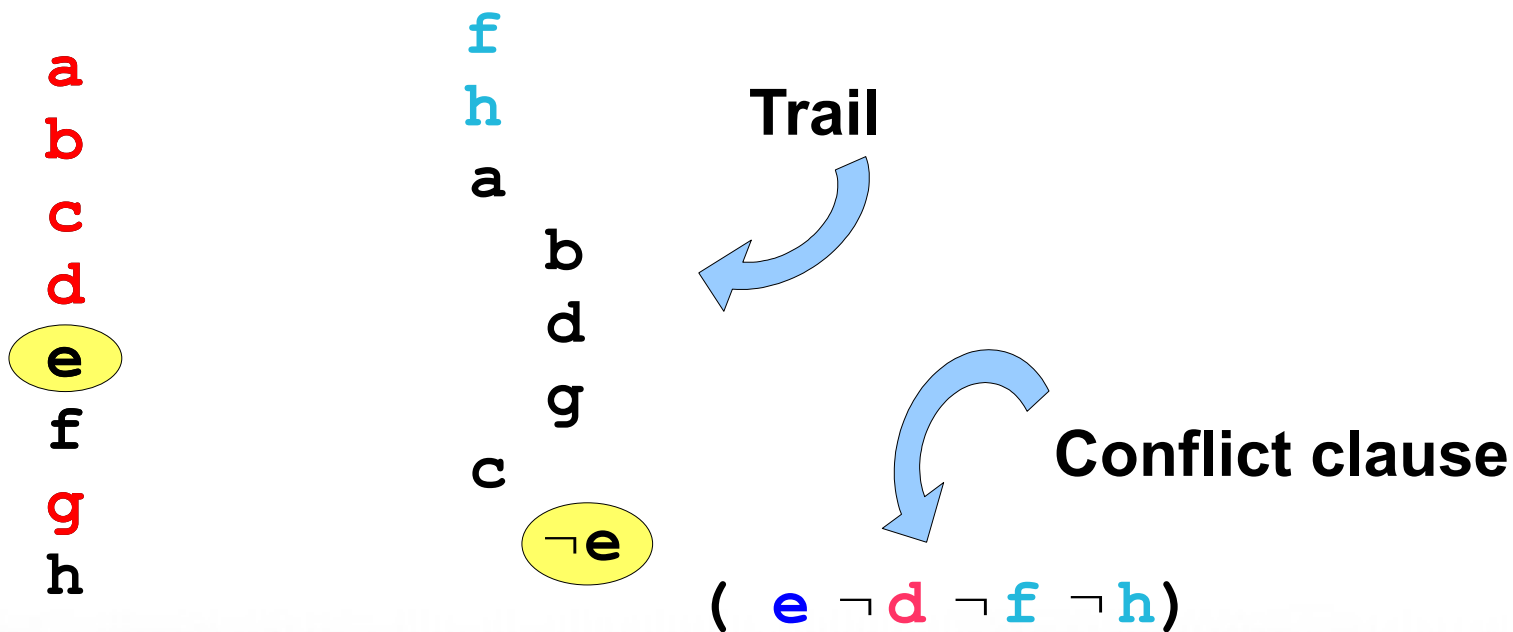
Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”



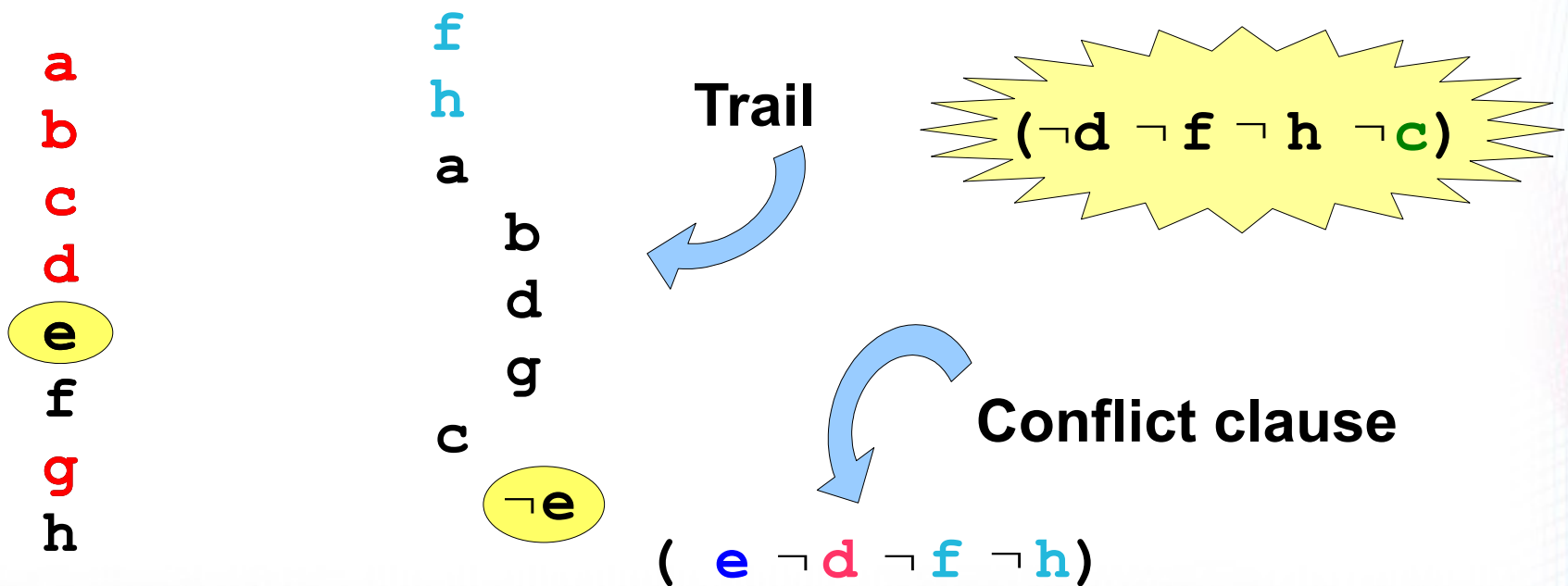
Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”



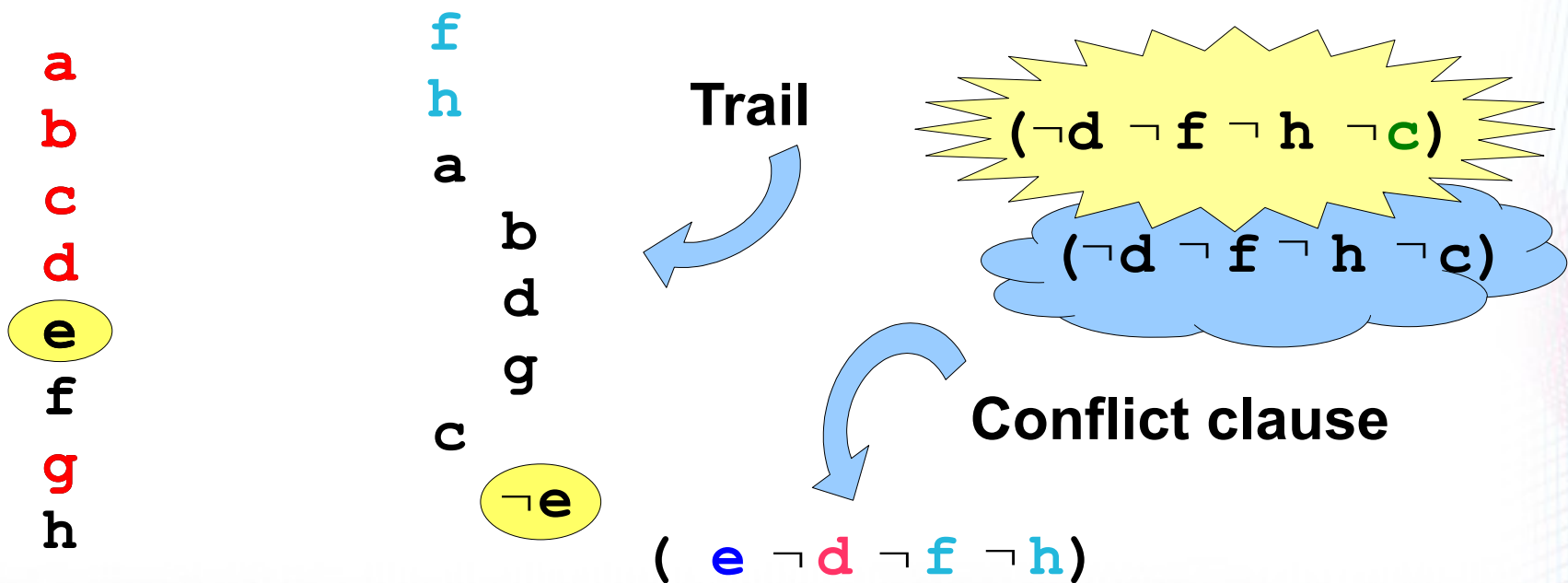
Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”



Local Search

- For better clauses, want an implication graph.
- How to generate it from existing assignment I?
 - CDLS: “simulate UP until it first deviates from I”



Local Search

- Problem: this is not a real conflict.
 - A SAT solver would not have learned here.

a
b
c
d
e
f
g
h

a
b
d
g
c
¬e

Local Search

- Problem: this is not a real conflict.
 - A SAT solver would not have learned here.
- You cannot “create” a learnable conflict from an arbitrary assignment

a
b
c
d
e
f
g
h

a
b
d
g
c
¬e

“Unlearnable” assignment

(a, b)
(c, d)
($\neg a$, c)
($\neg b$, d)
($\neg c$, a)
($\neg d$, b)

$\neg a \quad \neg b \quad \neg c \quad \neg d$

“Unlearnable” assignment

(a, b)

(c, d)

($\neg a$, c)

($\neg b$, d)

($\neg c$, a)

($\neg d$, b)

$\neg a \quad \neg b \quad \neg c \quad \neg d$

$\neg a$

“Unlearnable” assignment

(a, b)

(c, d)

($\neg a$, c)

($\neg b$, d)

($\neg c$, a)

($\neg d$, b)

$\neg a \quad \neg b \quad \neg c \quad \neg d$

$\neg a$

$\neg c$

b

“Unlearnable” assignment

(a, b)

(c, d)

($\neg a$, c)

($\neg b$, d)

($\neg c$, a)

($\neg d$, b)

$\neg a \quad \neg b \quad \neg c \quad \neg d$

$\neg a$

$\neg c$

b

“Unlearnable” assignment

(a, b)

(c, d)

($\neg a$, c)

($\neg b$, d)

($\neg c$, a)

($\neg d$, b)

$\neg a$ $\neg b$ $\neg c$ $\neg d$

$\neg a$

$\neg b$

$\neg c$

$\neg d$

$\neg c$

$\neg d$

$\neg a$

$\neg b$

b

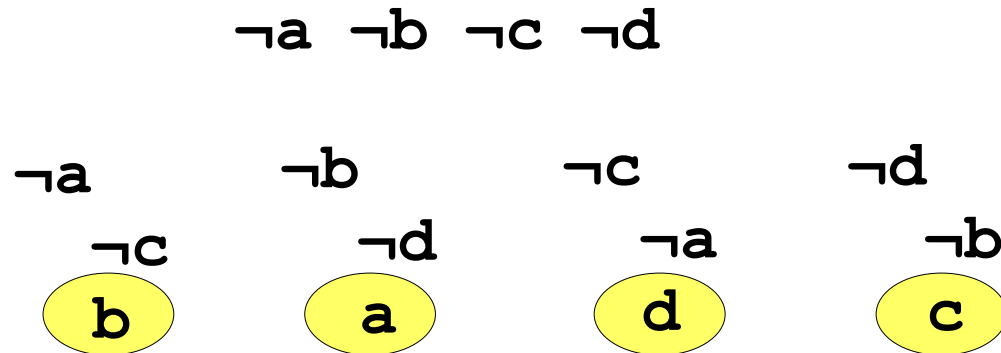
a

d

c

“Unlearnable” assignment

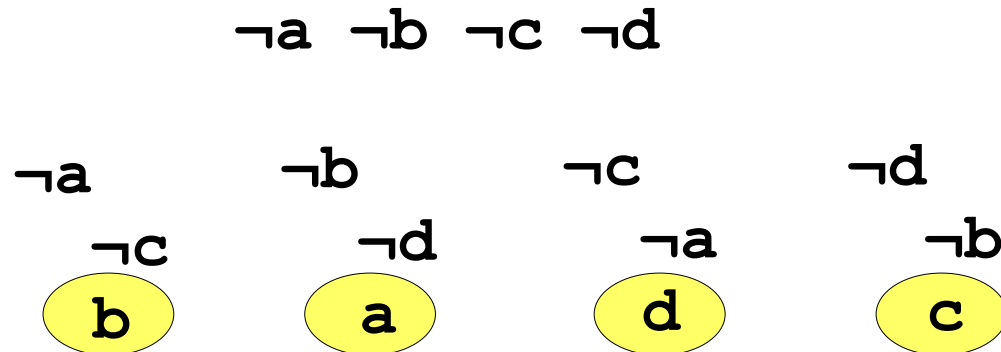
(a, b)
(c, d)
($\neg a$, c)
($\neg b$, d)
($\neg c$, a)
($\neg d$, b)



Can we go around this limitation?

“Unlearnable” assignment

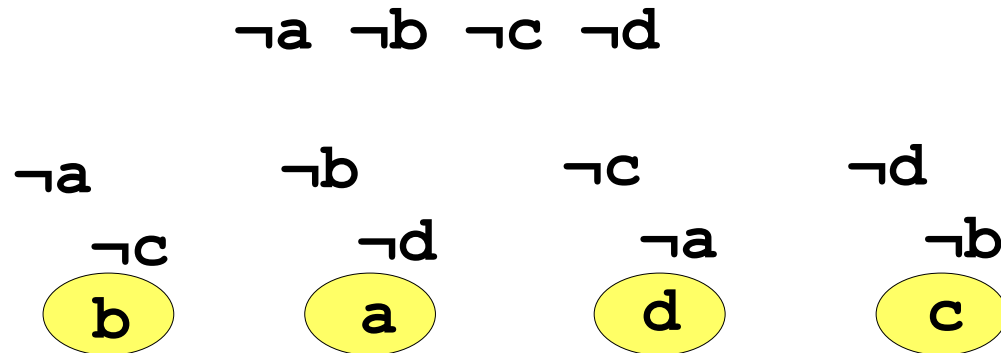
(a, b)
(c, d)
($\neg a$, c)
($\neg b$, d)
($\neg c$, a)
($\neg d$, b)



Can we go around this limitation?
• Allow new values?

“Unlearnable” assignment

(a, b)
(c, d)
($\neg a$, c)
($\neg b$, d)
($\neg c$, a)
($\neg d$, b)



Can we go around this limitation?

- Allow new values?
- Ignore unneeded implications?

“Unlearnable” assignment

(a, b)

(c, d)

($\neg a$, c)

($\neg b$, d)

($\neg c$, a)

($\neg d$, b)

$\neg a$ $\neg b$ $\neg c$ $\neg d$

$\neg a$

$\neg b$

$\neg c$

$\neg d$

$\neg c$

$\neg d$

$\neg a$

$\neg b$

b

a

d

c

$\neg b$

$\neg d$

a

“Unlearnable” assignment

● (a, b)

● (c, d)

($\neg a$, c)

($\neg b$, d)

($\neg c$, a)

($\neg d$, b)

$\neg a$ $\neg b$ $\neg c$ $\neg d$

$\neg a$

$\neg b$

$\neg c$

$\neg d$

$\neg c$

$\neg d$

$\neg a$

$\neg b$

b

a

d

c

$\neg b$

$\neg d$

a

“Unlearnable” assignment

● (a, b)

● (c, d)

($\neg a$, c)

($\neg b$, d)

($\neg c$, a)

($\neg d$, b)

$\neg a$ $\neg b$ $\neg c$ $\neg d$

$\neg a$

$\neg b$

$\neg c$

$\neg d$

$\neg c$

$\neg d$

$\neg a$

$\neg b$

b

a

d

c

$\neg b$

$\neg d$

a

- So: the problem is in the heuristic.
- We don't want “unlearnable” assignments to be a local minima
- *Note:* above assignment is a local minima for clause counting

choosing variables to flip through CDCL

- Inspired by intuition from CDCL solvers
- Uses a base heuristic for variable ordering
(For eg, VSIDS)
- Guarantees a learnable clause
- Makes the local search solver behave exactly as a CDCL solver

CDCL Local Search

`pickVar(I)`

- Order I in a UP-compatible order
 - implied variables have to appear as soon as possible after their reasons
 - (see the paper for more precise definition)
- Pick a false clause C that becomes unit earliest
 - It would have implied some variable v
- If $\neg v$ is implied earlier, have a real conflict.
 - learn a clause from C using the above ordering
- Otherwise, flip v .

Exploiting the insight

- A more efficient way to select variables to flip
 - Vector or vectors instead of a trail?
 - Completely different implementation?
- More freedom
 - Generating other empowering clauses and/or multiple ones from the same assignment?

Learning multiple clauses at once

- Preliminary work
 - currently no way to estimate which are “good” clauses in this context
 - doesn't provide empowerment guarantees
- First conflict clause at every decision level
 - if α does not derives any failed implications, “asserting” clause $(\alpha \rightarrow y)$ is empowering

Learning multiple clauses at once

Number of SAT11 problems solved within 1000s: **All**, True, False instances

Family	Minisat	C_1	C_5	C_10	C_100	C_1000	C_All
	A S U	A S U	A S U	A S U	A S U	A S U	A S U
fuhs (34)	10 9 1	10 8 2	9 8 1	9 8 1	10 7 3	10 8 2	7 7 0
manthey (9)	3 3 0	2 2 0	2 2 0	3 3 0	3 3 0	1 1 0	0 0 0
jarvisalo (47)	24 8 16	24 8 16	24 9 15	24 10 14	25 9 16	17 6 11	15 6 9
leberre (17)	11 4 7	13 6 7	14 6 8	13 6 7	13 6 7	12 5 7	7 1 6
rintanen (30)	9 7 2	8 5 3	7 4 3	8 5 3	8 5 3	2 1 1	1 0 1
kullmann (13)	2 2 0	3 3 0	3 3 0	2 2 0	4 4 0	3 3 0	3 3 0
Total (150)	59 33 26	60 32 28	59 32 27	59 34 25	63 34 29	45 24 21	33 17 16

- **C_All** learns all clauses all the time
- **C_n** learns all clauses n times every restart, standard learning at other times

Future work

- A more efficient way to calculate the heuristic
- More freedom
 - Generating other empowering clauses and/or multiple ones from the same assignment?
- Relaxing the restrictions in QBF?
- Other local search heuristics that guarantee “real” conflicts?
- Local search “base” heuristics?

Conclusion

- Reformulated CDCL algorithm as local search
- The trail is simply an efficient way for computing the heuristic
- New view to be explored, and more flexibility to be investigated
 - learning other clauses?
 - better ways of computing the heuristic?
 - flexibility for QBF?