Beyond Unit Propagation in SAT Solving

Michael Kaufmann and Stephan Kottler

University of Tuebingen

SEA 2011

10th International Symposium on Experimental Algorithms
May 5 - 7, 2011 - Orthodox Academy of Crete, Kolimpari Chania, Greece

WHY SAT?

Bounded Model Checking

Verification

Plugin System

Automotive Product Configuration

Why SAT?

Bounded Model Checking

Verification

Plugin System

Automotive Product Configuration

SAT-Solver

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

OUTLINE

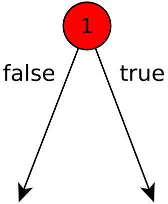
SAT Solving - DPLL

$$
\begin{aligned}
C_1 &= \{\overline{l_1} \vee l_6\} \\
C_2 &= \{\overline{l_6} \vee l_4\} \\
C_3 &= \{\overline{l_4} \vee \overline{l_6} \vee l_3\} \\
C_4 &= \{\overline{l_2} \vee l_7\}
\end{aligned}
$$

SAT Solving - DPLL

$$C_1 = \{\overline{l_1} \vee l_6\}$$
$$C_2 = \{\overline{l_6} \vee l_4\}$$
$$C_3 = \{\overline{l_4} \vee \overline{l_6} \vee l_3\}$$
$$C_4 = \{\overline{l_2} \vee l_7\}$$

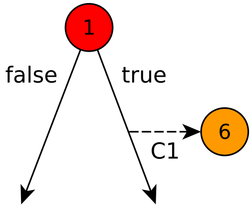
- Decisions



SAT SOLVING - DPLL

$C_1 = \{\overline{l_1} \vee l_6\}$
$C_2 = \{\overline{l_6} \vee l_4\}$
$C_3 = \{\overline{l_4} \vee \overline{l_6} \vee l_3\}$
$C_4 = \{\overline{l_2} \vee l_7\}$

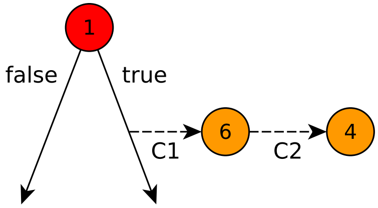
- Decisions
- Propagation of assignments



SAT Solving - DPLL

$C_1 = \{\overline{l_1} \vee l_6\}$
$C_2 = \{\overline{l_6} \vee l_4\}$
$C_3 = \{\overline{l_4} \vee \overline{l_6} \vee l_3\}$
$C_4 = \{\overline{l_2} \vee l_7\}$

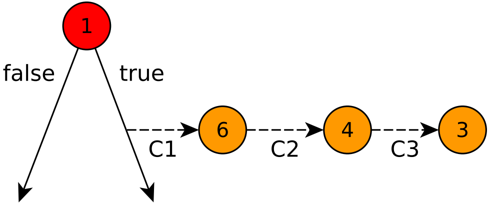
- Decisions
- Propagation of assignments



SAT SOLVING - DPLL

$C_1 = \{\overline{l_1} \vee l_6\}$
$C_2 = \{\overline{l_6} \vee l_4\}$
$C_3 = \{\overline{l_4} \vee \overline{l_6} \vee l_3\}$
$C_4 = \{\overline{l_2} \vee l_7\}$

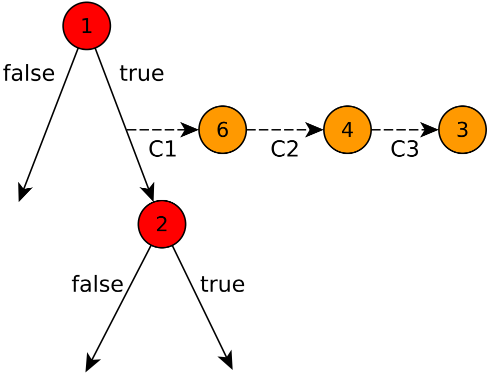
- Decisions
- Propagation of assignments



SAT Solving - DPLL

$$C_1 = \{\overline{l_1} \vee l_6\}$$
$$C_2 = \{\overline{l_6} \vee l_4\}$$
$$C_3 = \{\overline{l_4} \vee \overline{l_6} \vee l_3\}$$
$$C_4 = \{\overline{l_2} \vee l_7\}$$

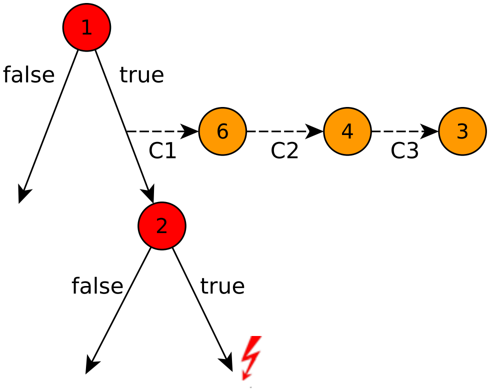
- Decisions
- Propagation of assignments



SAT SOLVING - DPLL

$C_1 = \{\overline{l_1} \vee l_6\}$
$C_2 = \{\overline{l_6} \vee l_4\}$
$C_3 = \{\overline{l_4} \vee \overline{l_6} \vee l_3\}$
$C_4 = \{\overline{l_2} \vee l_7\}$

- Decisions
- Propagation of assignments
- Conflict analysis



Boolean Constraint Propagation

- search constitutes partial assignment π
- consider clauses that are unit under π

BOOLEAN CONSTRAINT PROPAGATION

- search constitutes partial assignment π
- consider clauses that are unit under π

EXAMPLE (UNIT PROPAGATION)

$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$

$C = \{l_4 \lor l_5 \lor l_8\}$ is *unit under* $\pi \Rightarrow l_8$ is implied

Boolean Constraint Propagation

- search constitutes partial assignment π
- consider clauses that are unit under π

Example (Unit Propagation)

$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$

$C = \{l_4 \vee l_5 \vee l_8\}$ is *unit under* $\pi \Rightarrow l_8$ is implied

- very efficient implementations
- $\geq 80\%$ of runtime

IDEA: DON'T WAIT FOR UNITS

EXAMPLE

$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$

$C = \{l_4 \vee l_5 \vee l_1 \vee l_2 \vee l_3\}$

IDEA: DON'T WAIT FOR UNITS

EXAMPLE

$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$

$C = \{l_4 \vee l_5 \vee l_1 \vee l_2 \vee l_3\}$

What can we do?

IDEA: DON'T WAIT FOR UNITS

EXAMPLE

$\pi = \overline{l_4}, \overline{l_5}, l_6 \dots$
$C = \{l_4 \vee l_5 \vee l_1 \vee l_2 \vee l_3\}$

Clearly: one of l_1, l_2, l_3 has to be assigned

IDEA: DON'T WAIT FOR UNITS

EXAMPLE

$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$
$C = \{l_4 \vee l_5 \vee l_1 \vee l_2 \vee l_3\}$

Clearly: one of l_1, l_2, l_3 has to be assigned

Might be that all unassigned literals have common *direct* implication:
e.g. $l_1 \Rightarrow l_7$, $l_2 \Rightarrow l_7$, $l_3 \Rightarrow l_7$

IDEA: DON'T WAIT FOR UNITS

EXAMPLE

$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$
$C = \{l_4 \vee l_5 \vee l_1 \vee l_2 \vee l_3\}$

Clearly: one of l_1, l_2, l_3 has to be assigned

Might be that all unassigned literals have common *direct* implication:
e.g. $l_1 \Rightarrow l_7$, $l_2 \Rightarrow l_7$, $l_3 \Rightarrow l_7$

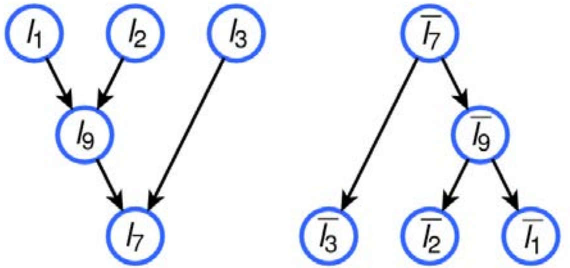
l_7 can already be assigned!

Direct implications in CNF

$$
\begin{aligned}
C_1 &= \{\overline{l_1} \vee l_9\} \\
C_2 &= \{\overline{l_2} \vee l_9\} \\
C_3 &= \{\overline{l_3} \vee l_7\} \\
C_4 &= \{\overline{l_9} \vee l_7\}
\end{aligned}
$$

DIRECT IMPLICATIONS IN CNF

$$C_1 = \{\overline{l_1} \vee l_9\}$$
$$C_2 = \{\overline{l_2} \vee l_9\}$$
$$C_3 = \{\overline{l_3} \vee l_7\}$$
$$C_4 = \{\overline{l_9} \vee l_7\}$$



Implication graph induced by binary clauses

JUST A GRAPH PROBLEM

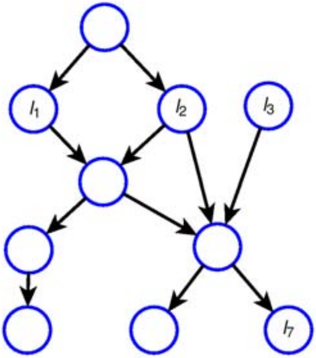
QUESTION?

Is there a common successor for a set of vertices?

JUST A GRAPH PROBLEM

QUESTION?

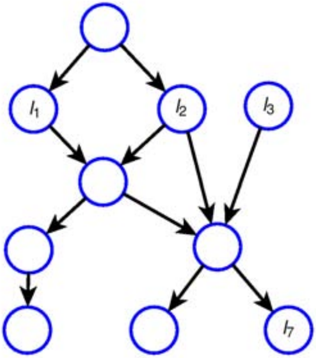
Is there a common successor for a set of vertices?



Just a graph problem

Question?

Is there a common successor for a set of vertices?



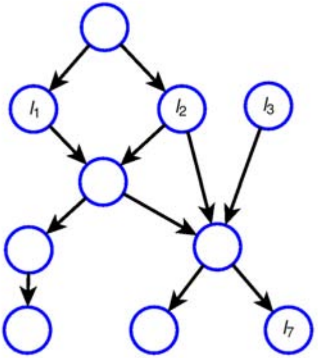
Trivial approach

Keep one bit for each pair of literals (l_i, l_j) which is set if l_i and l_j have a common successor

JUST A GRAPH PROBLEM

QUESTION?

Is there a common successor for a set of vertices?



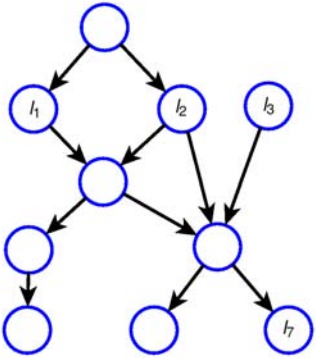
TRIVIAL APPROACH

Keep one bit for each pair of literals (l_i, l_j) which is set if l_i and l_j have a common successor

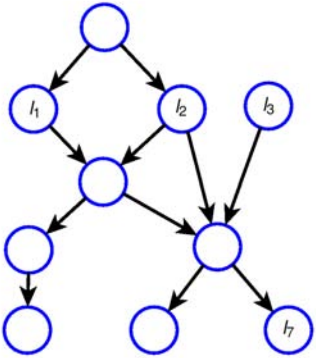
$\Downarrow$

Much too big!!

MATRIX COMPRESSION



MATRIX COMPRESSION

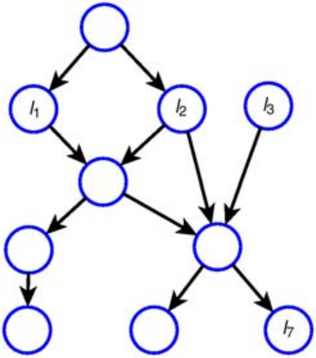


ONE IDEA:

In a DAG two vertices have common successor iff they reach same sink

$\Rightarrow$ store reachability of sinks

MATRIX COMPRESSION



ONE IDEA:

In a DAG two vertices have common successor iff they reach same sink

$\Rightarrow$ store reachability of sinks

...

...more compression techniques to make it work! [see paper!]

REVIEW

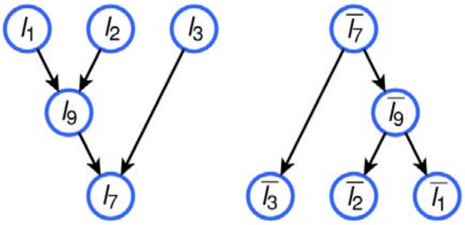
- matrices are still too big for some SAT instances
- adding many binary clauses requires matrix updates
- quite some work for implementation

S\ I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	x							x									x	
2									x									
3									x									
4																	x	
5	x													x				
6			x															
7					x													
8		x											x			x		
9			x			x		x		x				x				
10	x																x	
11		x				x	x	x										
12					x							x	x			x		
13				x		x				x	x							
14	x		x														x	
15														x				
16		x					x										x	
17				x		x				x	x				x			
18																		x
19	x										x						x	x
20															x			
21				x								x				x	x	x

SINKS AND ROOTS

COMPLEMENTARY COMPONENTS

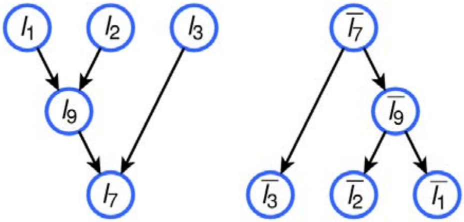
Flipped sinks of one component are roots in complementary component.



Sinks and roots

Complementary components

Flipped sinks of one component are roots in complementary component.



Still valid if complementary components are connected!

COLLECTING SINK TAGS

IDEA

Collect and cache information during normal unit propagation of binary clauses.

$$C_1 = \{\overline{l_1} \vee l_9\}$$
$$C_2 = \{l_2 \vee l_9\}$$
$$C_3 = \{l_3 \vee l_7\}$$
$$C_4 = \{l_9 \vee l_7\}$$

$\overline{l_7}$

Tag Table:

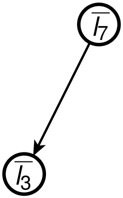
l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
↓	↓	↓	↓	↓	↓	↓	↓	↓
-	-	-	-	-	-	-	-	-

Collecting sink tags

Idea

Collect and cache information during normal unit propagation of binary clauses.

$C_1 = \{\overline{l_1} \vee l_9\}$
$C_2 = \{\overline{l_2} \vee l_9\}$
$C_3 = \{\overline{l_3} \vee l_7\}$
$C_4 = \{\overline{l_9} \vee l_7\}$



Tag Table:

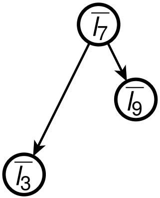
l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
-	-	l_7	-	-	-	-	-	-

Collecting sink tags

Idea

Collect and cache information during normal unit propagation of binary clauses.

$C_1 = \{\overline{l_1} \vee l_9\}$
$C_2 = \{\overline{l_2} \vee l_9\}$
$C_3 = \{\overline{l_3} \vee l_7\}$
$C_4 = \{\overline{l_9} \vee l_7\}$



Tag Table:

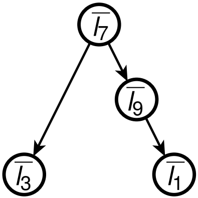
l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
↓	↓	↓	↓	↓	↓	↓	↓	↓
-	-	l_7	-	-	-	-	-	l_7

COLLECTING SINK TAGS

IDEA

Collect and cache information during normal unit propagation of binary clauses.

$$C_1 = \{\overline{l_1} \vee l_9\}$$
$$C_2 = \{\overline{l_2} \vee l_9\}$$
$$C_3 = \{\overline{l_3} \vee l_7\}$$
$$C_4 = \{\overline{l_9} \vee l_7\}$$



Tag Table:

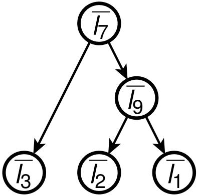
l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
l_7	-	l_7	-	-	-	-	-	l_7

COLLECTING SINK TAGS

IDEA

Collect and cache information during normal unit propagation of binary clauses.

$C_1 = \{\overline{l_1} \vee l_9\}$
$C_2 = \{\overline{l_2} \vee l_9\}$
$C_3 = \{\overline{l_3} \vee l_7\}$
$C_4 = \{\overline{l_9} \vee l_7\}$



Tag Table:

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
↓	↓	↓	↓	↓	↓	↓	↓	↓
l_7	l_7	l_7	-	-	-	-	-	l_7

Using sink tags

Tag Table:

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
l_7	l_7	l_7	-	-	-	-	-	l_7

Using sink tags

Tag Table:

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
l_7	l_7	l_7	-	-	-	-	-	l_7

Example

$$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$$
$$C = \{l_4 \vee l_5 \vee l_1 \vee l_2 \vee l_3\}$$

What can we do?

Using sink tags

Tag Table:

l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
l_7	l_7	l_7	-	-	-	-	-	l_7

Example

$\pi = \overline{l_4}, \overline{l_5}, l_6 \ldots$
$C = \{l_4 \vee l_5 \vee l_1 \vee l_2 \vee l_3\}$

What can we do? $\Rightarrow$ Simple table lookup

MATRIX VS. TAGS

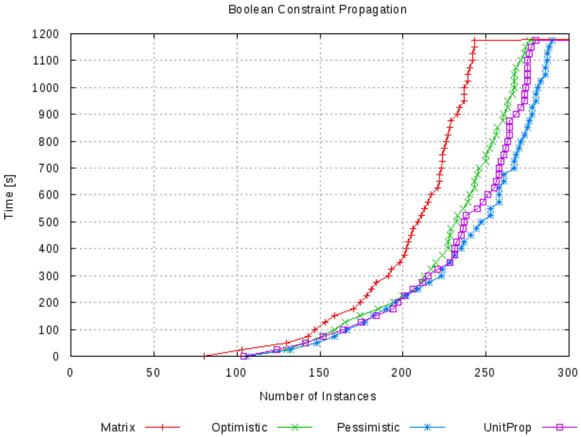
- Tests on 500 hard instances of previous SAT competitions
- Timeout for each instance 1200 seconds

MATRIX VS. TAGS

- Tests on 500 hard instances of previous SAT competitions
- Timeout for each instance 1200 seconds

	Matrix		Tags	
	avg	max	avg	max
ext. Prop / Decisions [%]	63.24	1581.93	33.71	1340.64
Implied Binaries	16816.36	235042	9100.49	152728
Implied Units	101.48	2722	146.71	4386

RUNTIME


Boolean Constraint Propagation
Time [s]
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
0
50
100
150
200
250
300
Number of Instances
Matrix
Optimistic
Pessimistic
UnitProp

CONCLUSION

- Analysed Boolean Constraint Propagation
- Most quality improvement with matrix approach $\rightarrow$ but bad runtime
- Tag approach still clearly better than Unit Propagation $\rightarrow$ comes for free!!

CONCLUSION

- Analysed Boolean Constraint Propagation
- Most quality improvement with matrix approach $\rightarrow$ but bad runtime
- Tag approach still clearly better than Unit Propagation $\rightarrow$ comes for free!!

Thank you!