Projected d-DNNF Compilation for Feature Models

Master's Thesis | Jacob Loth | November 18, 2023

universität ulm

1. Motivation

Model Counting

Problem

Counting the number of satisfiable assignments of a propositional formula F. Denoted as $|F|$.

p	q	$F = a \wedge b$
1	1	1
1	0	0
0	1	0
0	0	0

#SAT

Counts the number of solutions of a propositional formula.
Worst-case exponential complexity!

d-DNNF

Any propositional formula which is:

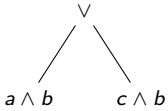
deterministic

Exclusive or-operators $F = A \vee B$

Never simultaneous $A = 1$ and $B = 1$

If-then-else

$|F| = |A| + |B|$



d-DNNF

Any propositional formula which is:

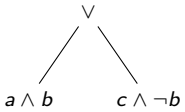
deterministic

Exclusive or-operators $F = A \vee B$

Never simultaneous $A = 1$ and $B = 1$

If-then-else

$|F| = |A| + |B|$



d-DNNF

Any propositional formula which is:

deterministic

Exclusive or-operators $F = A \vee B$

Never simultaneous $A = 1$ and $B = 1$

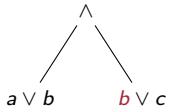
If-then-else

$|F| = |A| + |B|$

Decomposable

And-operands $F = A \wedge B$ never share variables

$|F| = |A| * |B|$



d-DNNF

Any propositional formula which is:

deterministic

Exclusive or-operators $F = A \vee B$

Never simultaneous $A = 1$ and $B = 1$

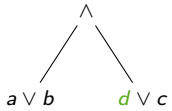
If-then-else

$|F| = |A| + |B|$

Decomposable

And-operands $F = A \wedge B$ never share variables

$|F| = |A| * |B|$



d-DNNF

Any propositional formula which is:

deterministic

Exclusive or-operators $F = A \vee B$

Never simultaneous $A = 1$ and $B = 1$

If-then-else

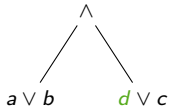
$|F| = |A| + |B|$

Decomposable

And-operands $F = A \wedge B$ never share variables

$|F| = |A| * |B|$

Negation **N**ormal **F**orm



d-DNNF

Any propositional formula which is:

deterministic

Exclusive or-operators $F = A \vee B$

Never simultaneous $A = 1$ and $B = 1$

If-then-else

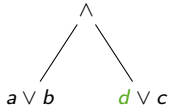
$|F| = |A| + |B|$

Decomposable

And-operands $F = A \wedge B$ never share variables

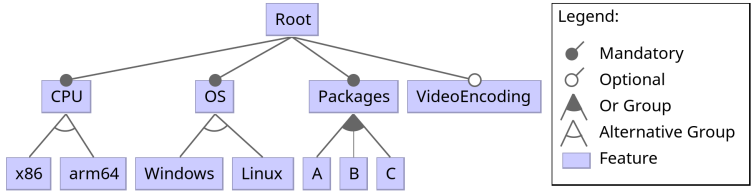
$|F| = |A| * |B|$

Negation **N**ormal **F**orm



d-DNNF formulas allow linear-time model counting
d-DNNF Compilation: CNF → d-DNNF
Knowledge-Compilation: It's still just a formula

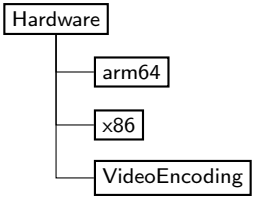
Feature Models

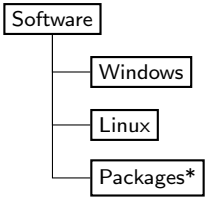


For this thesis, we can treat feature models as propositional formulas

Feature-Model Slicing

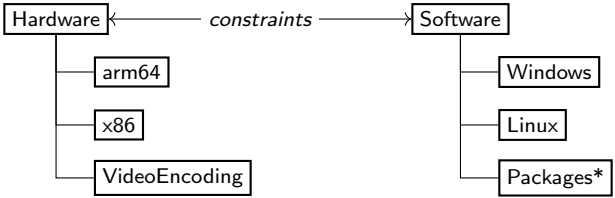
Feature Model





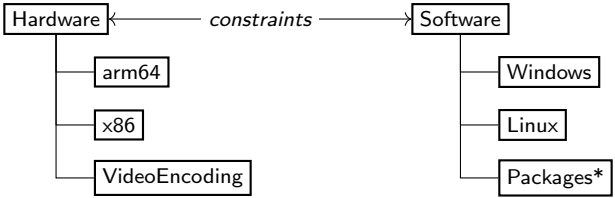
Feature-Model Slicing

Feature Model



Feature-Model Slicing

Feature Model

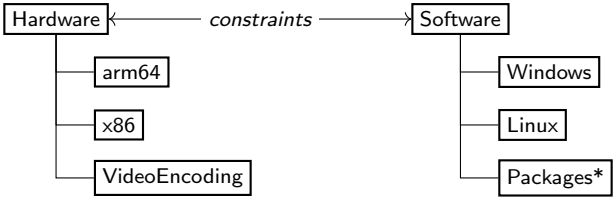


Problem

How many hardware configurations?

Feature-Model Slicing

Feature Model



Problem

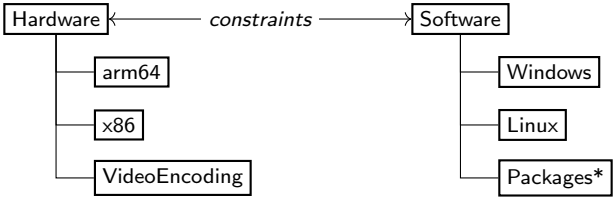
How many hardware configurations?

Transitive Constraints

$$\text{VideoEncoding} \implies \text{Windows}$$
$$\text{Windows} \implies \text{x86}$$

Feature-Model Slicing

Feature Model



Problem

How many hardware configurations?

Transitive Constraints

VideoEncoding $\Longrightarrow$ Windows

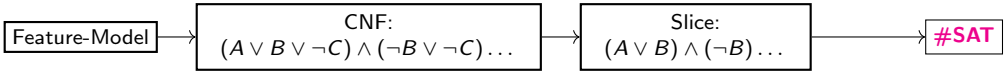
Windows $\Longrightarrow$ x86

Sliced: VideoEncoding $\Longrightarrow$ x86

Feature-Model Slicing

Problem

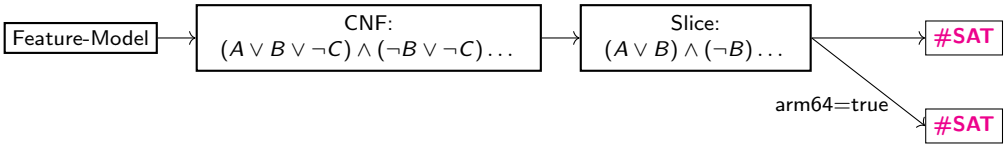
How many hardware configurations?



Feature-Model Slicing

Problem

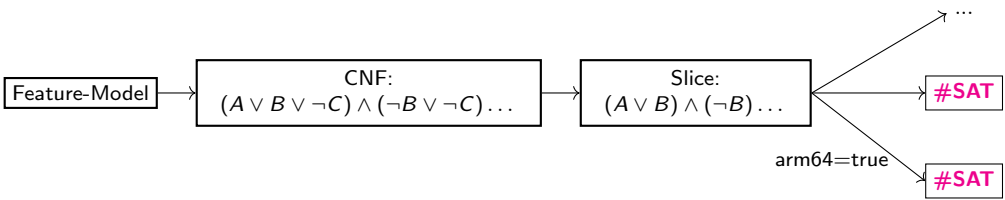
How many hardware configurations with arm64?



Feature-Model Slicing

Problem

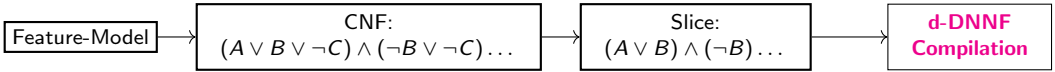
How many hardware configurations with X ?



Feature-Model Slicing

Problem

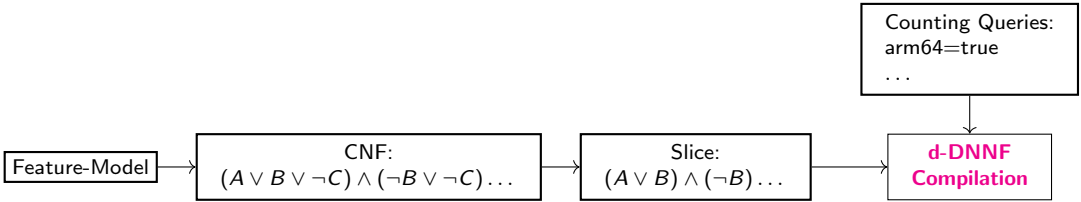
How many hardware configurations with X ?



Feature-Model Slicing

Problem

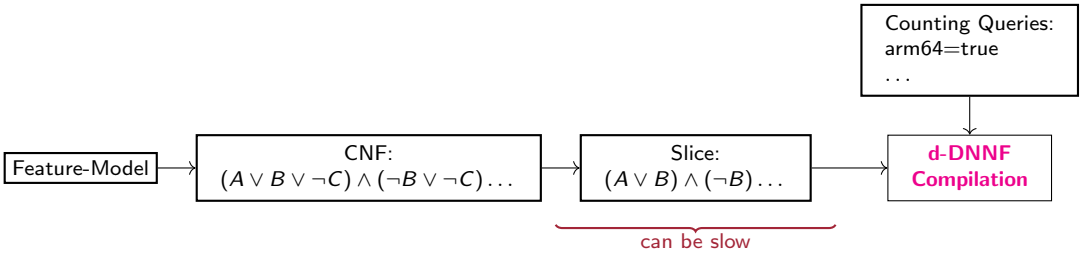
How many hardware configurations with X ?



Feature-Model Slicing

Problem

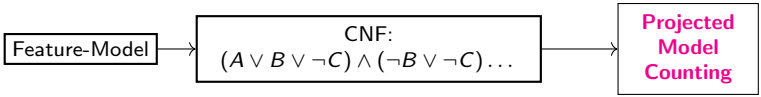
How many hardware configurations with X ?



Feature-Model Slicing

Problem

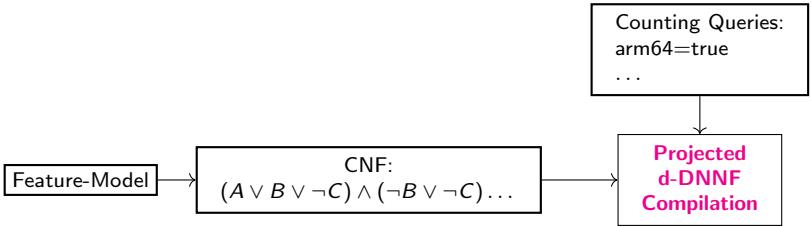
How many hardware configurations with X ?



Feature-Model Slicing

Problem

How many hardware configurations with X ?



Slicing Implementation[1]

Resolve all clauses with v with all clauses with $\neg v$

Resolving Two Clauses

$$(a_1 \vee a_2 \vee ... \vee v), (b_1 \vee b_2 \vee ... \vee \neg v) \rightarrow (a_1 \vee a_2 \vee ... \vee b_1 \vee b_2 \vee ...)$$

$$(\neg \mathit{VideoEncoding} \vee \mathit{Windows}), (x86 \vee \neg \mathit{Windows}) \rightarrow (\neg \mathit{VideoEncoding} \vee x86)$$

[1] Comparing Algorithms for Efficient Feature-Model Slicing, Krieter et al.

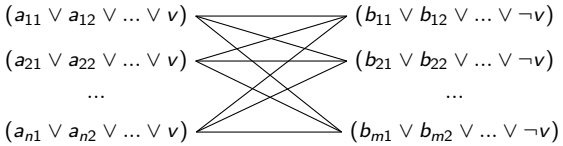
Slicing Implementation[1]

Resolve all clauses with v with all clauses with $\neg v$

Resolving Two Clauses

$$(a_1 \vee a_2 \vee ... \vee v), (b_1 \vee b_2 \vee ... \vee \neg v) \rightarrow (a_1 \vee a_2 \vee ... \vee b_1 \vee b_2 \vee ...)$$
$$(\neg \mathit{VideoEncoding} \vee \mathit{Windows}), (x86 \vee \neg \mathit{Windows}) \rightarrow (\neg \mathit{VideoEncoding} \vee x86)$$

Resolving Many Clauses



[1] Comparing Algorithms for Efficient Feature-Model Slicing, Krieter et al.

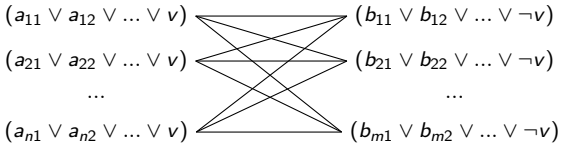
Slicing Implementation[1]

Resolve all clauses with v with all clauses with $\neg v$

Resolving Two Clauses

$$(a_1 \vee a_2 \vee ... \vee v), (b_1 \vee b_2 \vee ... \vee \neg v) \rightarrow (a_1 \vee a_2 \vee ... \vee b_1 \vee b_2 \vee ...)$$
$$(\neg VideoEncoding \vee Windows), (x86 \vee \neg Windows) \rightarrow (\neg VideoEncoding \vee x86)$$

Resolving Many Clauses



Exponential clause count increase for multiple variables.

[1] Comparing Algorithms for Efficient Feature-Model Slicing, Krieter et al.

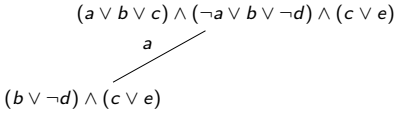
d-DNNF Compilation

DPLL

$$(a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e)$$

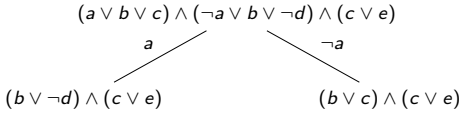
d-DNNF Compilation

DPLL



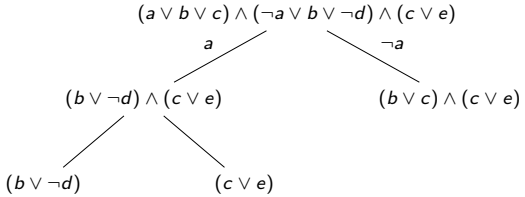
d-DNNF Compilation

DPLL



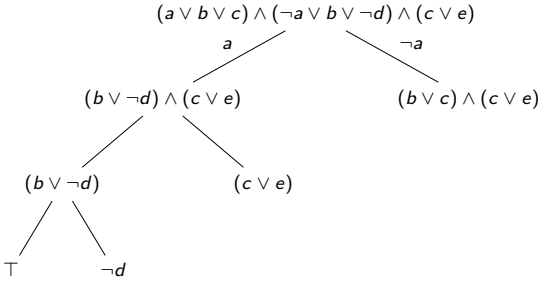
d-DNNF Compilation

DPLL



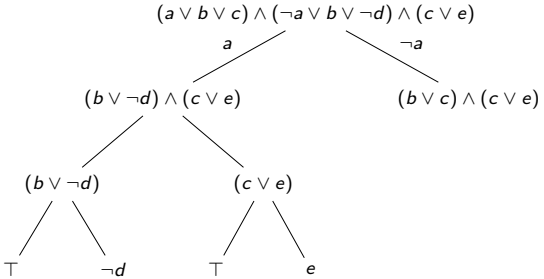
d-DNNF Compilation

DPLL



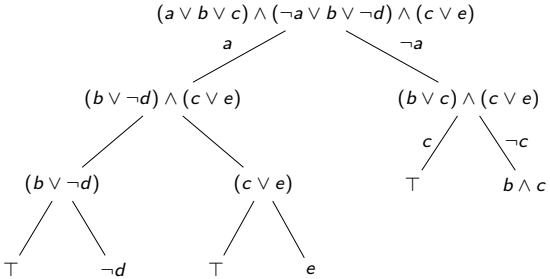
d-DNNF Compilation

DPLL



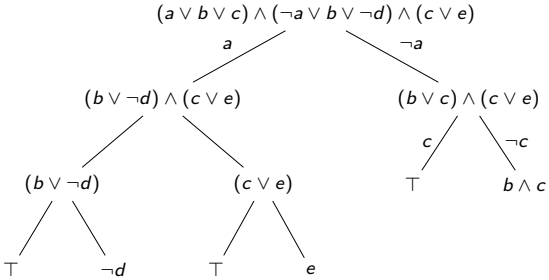
d-DNNF Compilation

DPLL

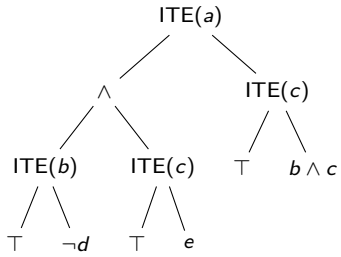


d-DNNF Compilation

DPLL



d-DNNF



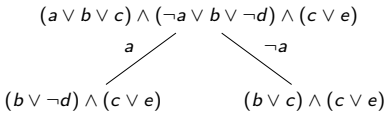
ITE = If Then Else

Variable Ordering

Vanilla DPLL has $O(2^n)$ runtime

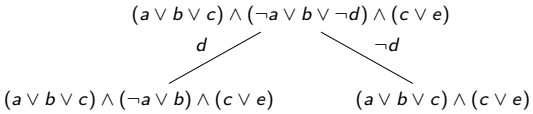
Variable Ordering

Vanilla DPLL has $O(2^n)$ runtime

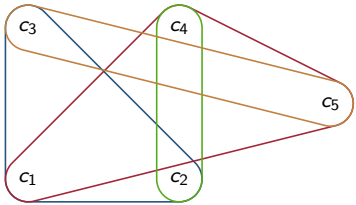


Variable Ordering

Vanilla DPLL has $O(2^n)$ runtime



Dual Hypergraph: CNF to Graph

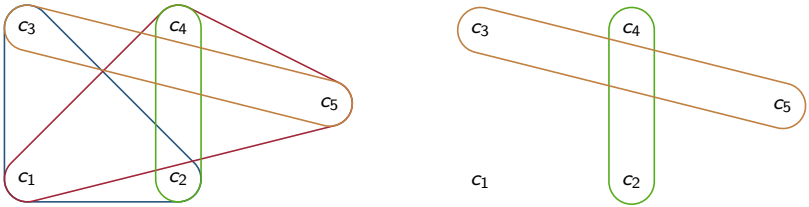


Construction

$$F = \underbrace{(a \vee b)}_{c_1} \wedge \underbrace{(a \vee \neg c)}_{c_2} \wedge \underbrace{(a \vee \neg d)}_{c_3} \wedge \underbrace{(b \vee \neg c)}_{c_4} \wedge \underbrace{(b \vee \neg d)}_{c_5}$$

Split formula into independent sub-problems of roughly equal size.

Dual Hypergraph: CNF to Graph



Construction

$$F = \underbrace{(a \vee b)}_{c_1} \wedge \underbrace{(a \vee \neg c)}_{c_2} \wedge \underbrace{(a \vee \neg d)}_{c_3} \wedge \underbrace{(b \vee \neg c)}_{c_4} \wedge \underbrace{(b \vee \neg d)}_{c_5}$$

Split formula into independent sub-problems of roughly equal size.

2. Our Contributions

Projected d-DNNF Compilation

Concept

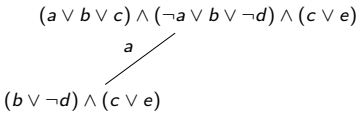
Projected variables $= a, b, d$
Sliced variables $= c, e$

$$(a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e)$$

Projected d-DNNF Compilation

Concept

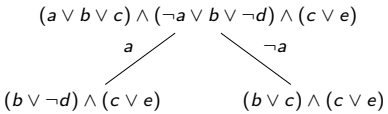
Projected variables $= a, b, d$
Sliced variables $= c, e$



Projected d-DNNF Compilation

Concept

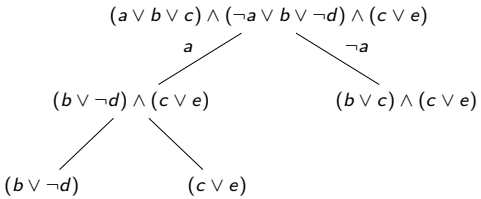
Projected variables $= a, b, d$
Sliced variables $= c, e$



Projected d-DNNF Compilation

Concept

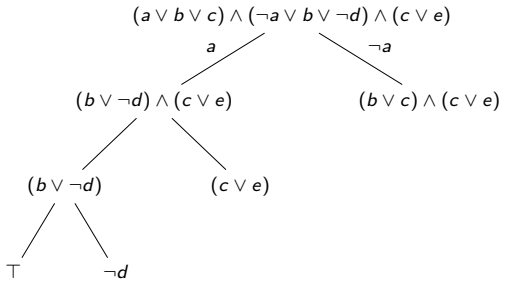
Projected variables $= a, b, d$
Sliced variables $= c, e$



Projected d-DNNF Compilation

Concept

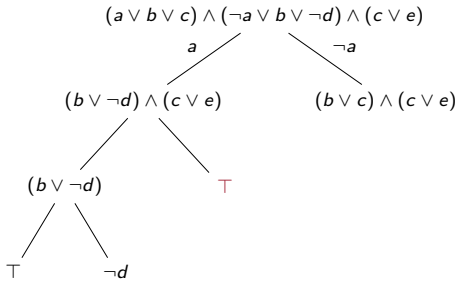
Projected variables $= a, b, d$
Sliced variables $= c, e$



Projected d-DNNF Compilation

Concept

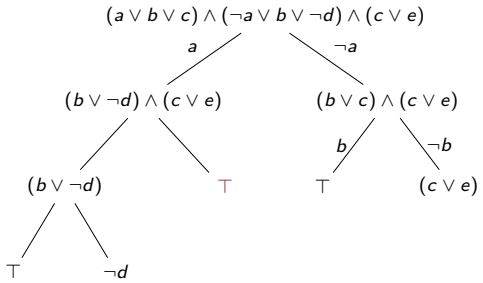
Projected variables $= a, b, d$
Sliced variables $= c, e$



Projected d-DNNF Compilation

Concept

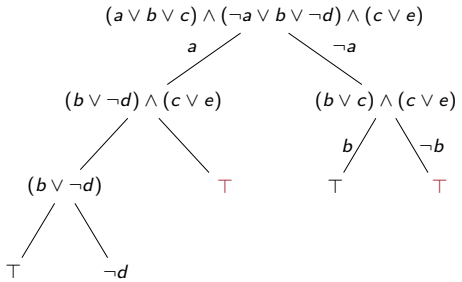
Projected variables $= a, b, d$
Sliced variables $= c, e$



Projected d-DNNF Compilation

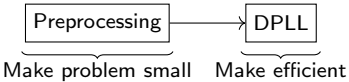
Concept

Projected variables $= a, b, d$
Sliced variables $= c, e$



Integration and Optimization in D4[3]

Two approaches: Better Preprocessing, Faster DPLL

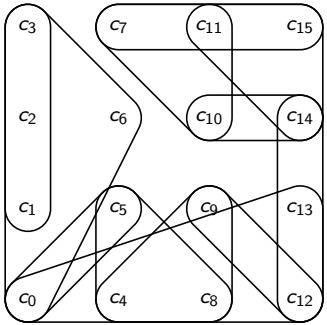


Contributions

- New dual weighted hypergraph partitioning heuristic
- Dynamic pure Literal elimination
- New partial resolution preprocessing heuristic based upon GPMC[2]
- Integration of multiple established preprocessing methods

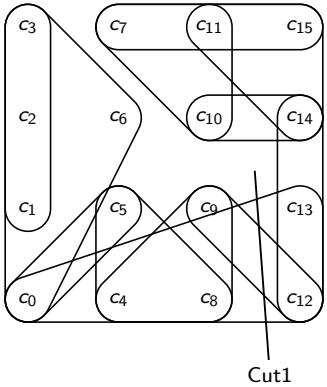
[2] https://git.trs.css.i.nagoya-u.ac.jp/k-hasimt/GPMC
[3] An Improved Decision-DNNF Compiler, Lagniez et al.

Dual Weighted Hypergraph Partitioning



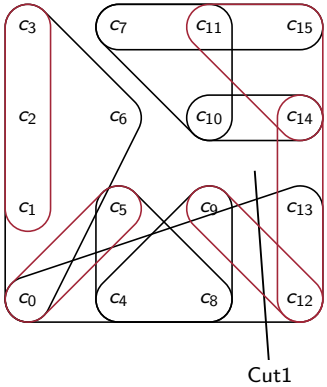
$F = c_0 \wedge c_1 \wedge ... \wedge c_{15}$

Dual Weighted Hypergraph Partitioning



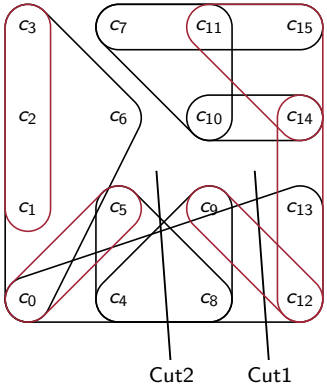
$F = c_0 \wedge c_1 \wedge ... \wedge c_{15}$

Dual Weighted Hypergraph Partitioning



$F = c_0 \wedge c_1 \wedge ... \wedge c_{15}$

Dual Weighted Hypergraph Partitioning



$F = c_0 \wedge c_1 \wedge ... \wedge c_{15}$

Partial Resolution

Greedily resolve "easy" sliced variables until the clause count increases

Partial Resolution

Greedily resolve "easy" sliced variables until the clause count increases

Clause Ratio

$F = (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e)$

Sliced variables $= c, e$

Resolution of c and e is trivial $\implies F = (\neg a \vee b \vee \neg d)$

Partial Resolution

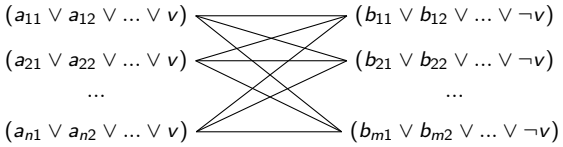
Greedily resolve "easy" sliced variables until the clause count increases

Clause Ratio

$F = (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e)$

Sliced variables $= c, e$

Resolution of c and e is trivial $\implies F = (\neg a \vee b \vee \neg d)$



Partial Resolution

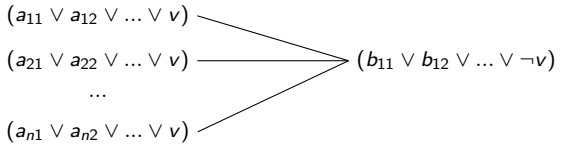
Greedily resolve "easy" sliced variables until the clause count increases

Clause Ratio

$F = (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e)$

Sliced variables $= c, e$

Resolution of c and e is trivial $\implies F = (\neg a \vee b \vee \neg d)$



Partial Resolution

Greedily resolve "easy" sliced variables until the clause count increases

Connectivity

$(\neg a \vee b \vee v), (a \vee c \vee d \vee \neg v) \rightarrow (\neg a \vee b \vee a \vee c) \equiv \top$

Partial Resolution

Greedily resolve "easy" sliced variables until the clause count increases

Connectivity

$(\neg a \vee b \vee v), (a \vee c \vee d \vee \neg v) \rightarrow (\neg a \vee b \vee a \vee c) \equiv \top$

Simpical Variable [1]: neighbors form a clique through clauses

[1] from GPMC source code

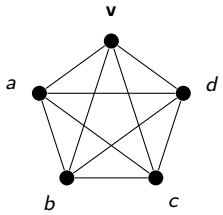
Partial Resolution

Greedily resolve "easy" sliced variables until the clause count increases

Connectivity

$(\neg a \vee b \vee v), (a \vee c \vee d \vee \neg v) \rightarrow (\neg a \vee b \vee a \vee c) \equiv \top$

Simpical Variable [1]: neighbors form a clique through clauses



Tree-likeness of Clauses $\cong$ Problem Complexity
Preserves tree-like structure

[1] from GPMC source code

Partial Resolution

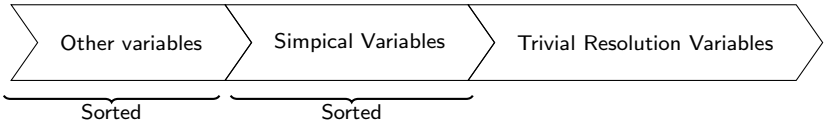
Greedily resolve "easy" sliced variables until the clause count increases

New Combined Heuristic

Group by:

1. Trivial resolution variables
2. Simpical variables
3. Other variables

Sort by: $v_p * v_n$ and average clause length

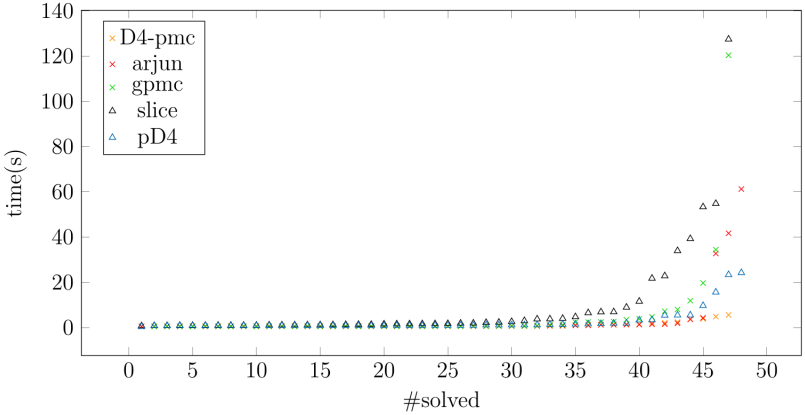


Results

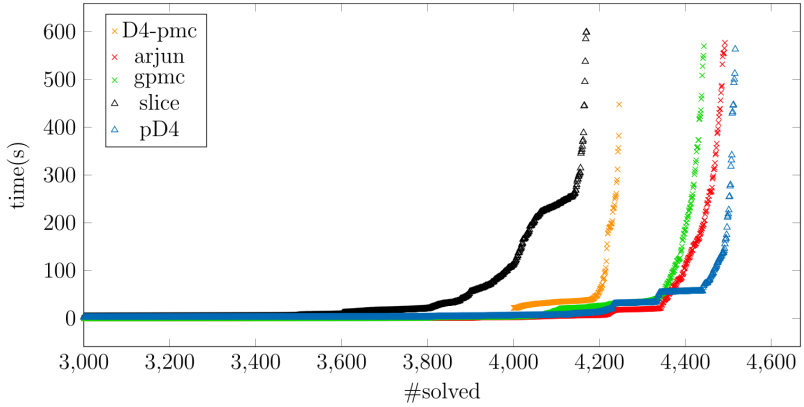
Solvers

- **pD4**: Our approach
- **slice**: Slicing followed by d-DNNF compilation
- **gpmc**: 1st place projected model counter in MC2022
- **D4-pmc**: 2nd place projected model counter in MC2022
- **arjun**: 3rd place projected model counter in MC2022

Experiment1: Industrial Projection



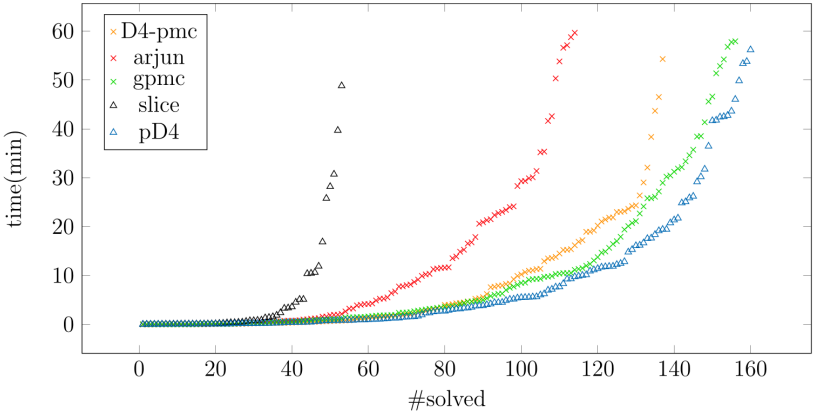
Experiment2: Generated Projection



Experiment2: Generated Projection

Model	d4-pmc	arjun	pD4	slice	gpmc
Smarch.2.6.32-2var	0	0	0	0	0
Smarch.2.6.28.6-icse11	0	0	0	0	0
Smarch.freetz	0	42	**65**	0	36
Smarch.buildroot	0	92	**100**	0	93
KConfig.linux-2.6.33.3	27	90	**96**	63	55
Smarch.embtoolkit	20	100	100	0	100
Smarch.freebsd-icse11	**100**	69	56	23	60
Smarch.uClinux-config	100	100	100	85	100
automotive02.automotive2_4	100	100	100	100	100

Experiment3: MC2022



Recap and Feature Work

Recap

- Combine slice and d-DNNF compilation

Recap and Feature Work

Recap

- Combine slice and d-DNNF compilation
- New heuristics for good performance

Recap and Feature Work

Recap

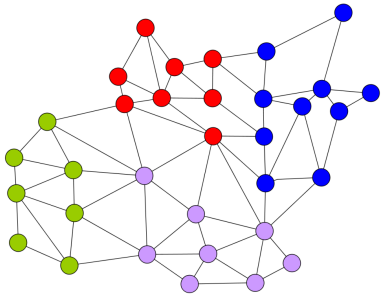
- Combine slice and d-DNNF compilation
- New heuristics for good performance
- Compile hard feature models like linux under projection

Recap and Feature Work

Recap

- Combine slice and d-DNNF compilation
- New heuristics for good performance
- Compile hard feature models like linux under projection
- Often comparable or smaller d-DNNF size

Recap and Feature Work



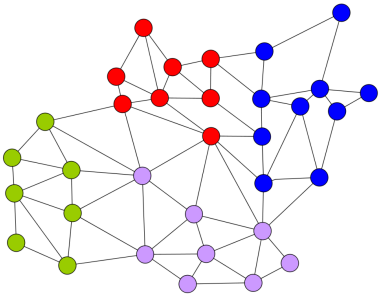
Recap

- Combine slice and d-DNNF compilation
- New heuristics for good performance
- Compile hard feature models like linux under projection
- Often comparable or smaller d-DNNF size

Future Work

Lots of new possible heuristics...

Recap and Feature Work



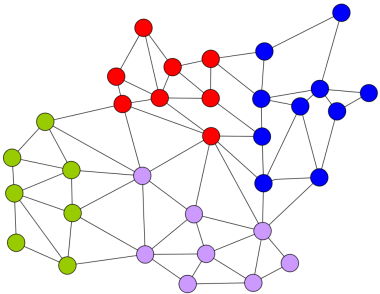
Recap

- Combine slice and d-DNNF compilation
- New heuristics for good performance
- Compile hard feature models like linux under projection
- Often comparable or smaller d-DNNF size

Future Work

Lots of new possible heuristics...
Hardware acceleration...

Recap and Feature Work



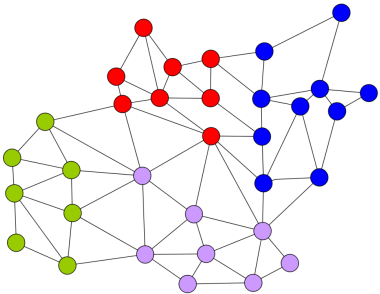
Recap

- Combine slice and d-DNNF compilation
- New heuristics for good performance
- Compile hard feature models like linux under projection
- Often comparable or smaller d-DNNF size

Future Work

Lots of new possible heuristics...
Hardware acceleration...
More applications for projected output d-DNNF

Recap and Feature Work



Recap

- Combine slice and d-DNNF compilation
- New heuristics for good performance
- Compile hard feature models like linux under projection
- Often comparable or smaller d-DNNF size

Future Work

Lots of new possible heuristics...
Hardware acceleration...
More applications for projected output d-DNNF