



Projected d-DNNF Compilation for Feature Models

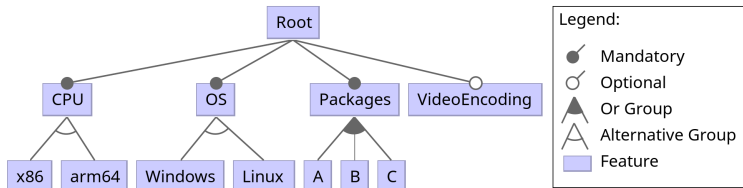
Master's Thesis | Jacob Loth | December 12, 2023



universität
uulm

1. Motivation

Feature Models

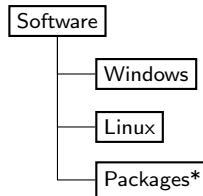
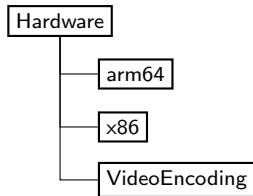


We can convert feature models to propositional formulas!

$$(x86 \wedge \neg arm64) \vee (\neg x86 \wedge arm64) \dots$$

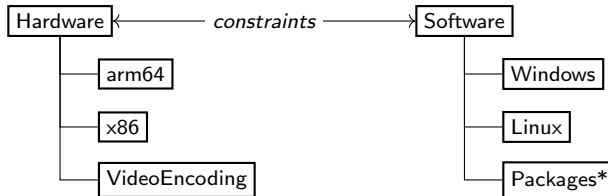
Feature-Model Slicing

Feature Model



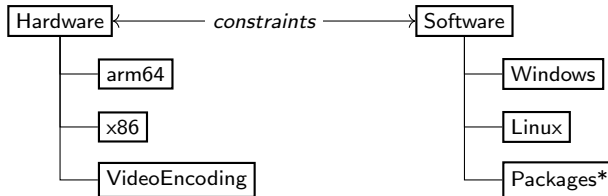
Feature-Model Slicing

Feature Model



Feature-Model Slicing

Feature Model

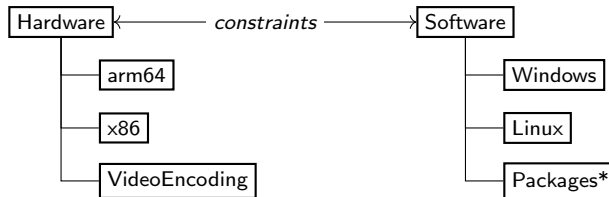


Problem

How many hardware configurations?

Feature-Model Slicing

Feature Model



Problem

How many hardware configurations?

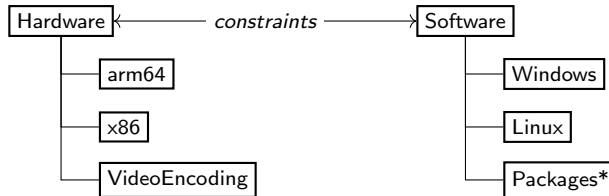
Transitive Constraints

VideoEncoding \Rightarrow Windows

Windows \Rightarrow x86

Feature-Model Slicing

Feature Model



Problem

How many hardware configurations?

Transitive Constraints

VideoEncoding \Rightarrow Windows
Windows \Rightarrow x86

Sliced: VideoEncoding \Rightarrow x86

Model Counting

Problem

- How many hardware configurations?

Model Counting

Problem

- How many hardware configurations?
- 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

Model Counting

Problem

- How many hardware configurations?
- 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:

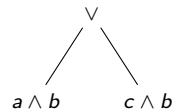
deterministic

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

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

If-then-else

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



Not a d-DNNF **✗**

d-DNNF

Any propositional formula which is:

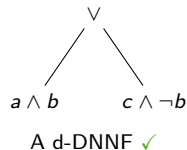
deterministic

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 $B = 1$

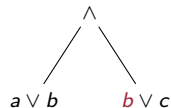
If-then-else

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

Decomposable

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

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



Not a d-DNNF **X**

d-DNNF

Any propositional formula which is:

deterministic

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

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

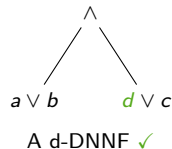
If-then-else

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

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

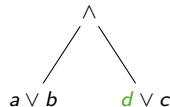
$|F| = |A| + |B|$

Decomposable

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

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

Negation Normal Form



d-DNNF

Any propositional formula which is:

deterministic

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

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

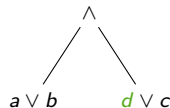
If-then-else

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

Decomposable

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

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



Negation Normal Form

d-DNNF formulas allow linear-time model counting

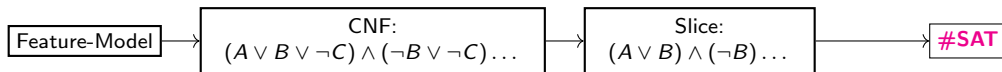
d-DNNF Compilation: CNF \rightarrow d-DNNF

Knowledge-Compilation: It's still just a formula

Feature-Model Slicing

Problem

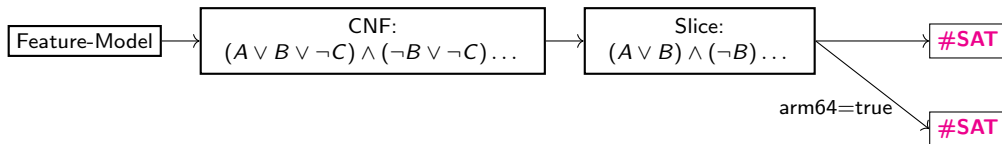
How many hardware configurations?



Feature-Model Slicing

Problem

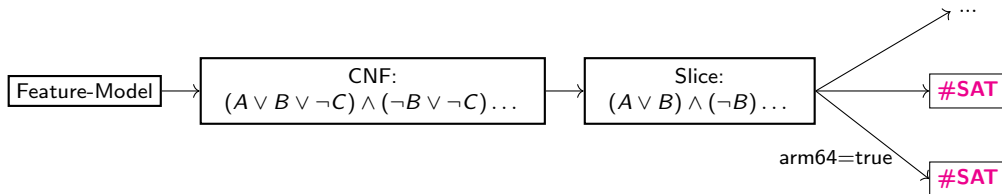
How many hardware configurations **with arm64**?



Feature-Model Slicing

Problem

How many hardware configurations **with X** ?



Feature-Model Slicing

Problem

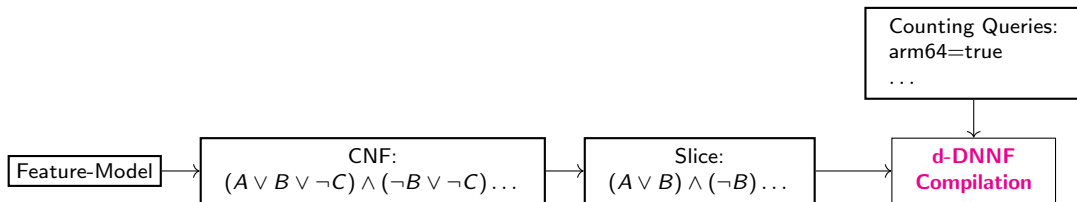
How many hardware configurations **with X** ?



Feature-Model Slicing

Problem

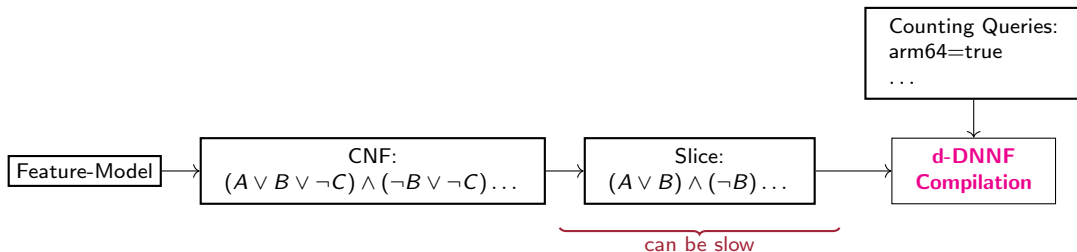
How many hardware configurations **with X** ?



Feature-Model Slicing

Problem

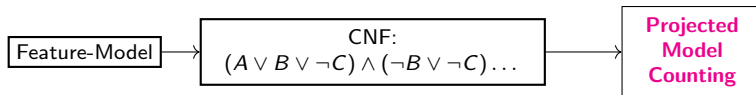
How many hardware configurations **with X** ?



Feature-Model Slicing

Problem

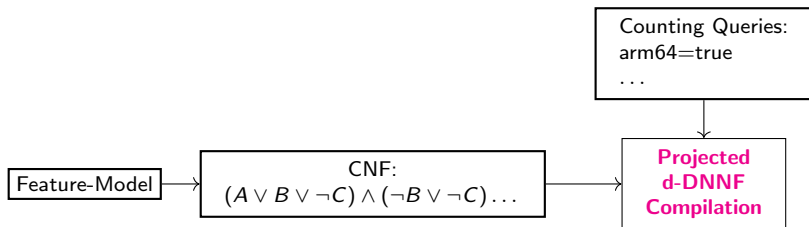
How many hardware configurations **with X** ?



Feature-Model Slicing

Problem

How many hardware configurations **with X** ?



Slicing Implementation¹

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

Resolving Two Clauses

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

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

Slicing Implementation¹

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

Resolving Two Clauses

$$\begin{aligned} &(\neg \text{VideoEncoding} \vee \text{Windows}), (\text{x86} \vee \neg \text{Windows}) \rightarrow (\neg \text{VideoEncoding} \vee \text{x86}) \\ &(a_1 \vee a_2 \vee \dots \vee v), (b_1 \vee b_2 \vee \dots \vee \neg v) \rightarrow (a_1 \vee a_2 \vee \dots \vee b_1 \vee b_2 \vee \dots) \end{aligned}$$

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

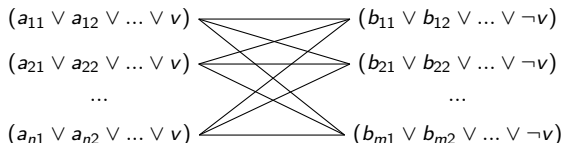
Slicing Implementation¹

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

Resolving Two Clauses

$$\begin{aligned} &(\neg \text{VideoEncoding} \vee \text{Windows}), (\text{x86} \vee \neg \text{Windows}) \rightarrow (\neg \text{VideoEncoding} \vee \text{x86}) \\ &(a_1 \vee a_2 \vee \dots \vee v), (b_1 \vee b_2 \vee \dots \vee \neg v) \rightarrow (a_1 \vee a_2 \vee \dots \vee b_1 \vee b_2 \vee \dots) \end{aligned}$$

Resolving Many Clauses



Exponential clause count increase for multiple variables.

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

d-DNNF Compilation

DPLL

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

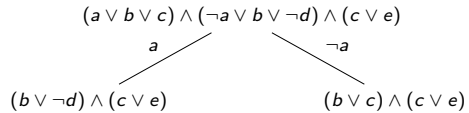
d-DNNF Compilation

DPLL

$$\begin{array}{c} (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e) \\ \swarrow a \\ (b \vee \neg d) \wedge (c \vee e) \end{array}$$

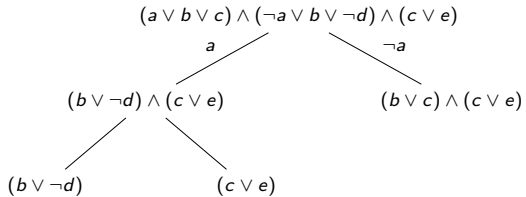
d-DNNF Compilation

DPLL



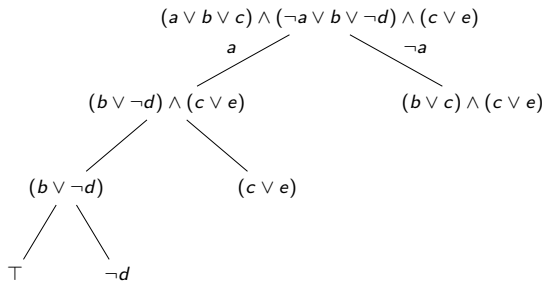
d-DNNF Compilation

DPLL



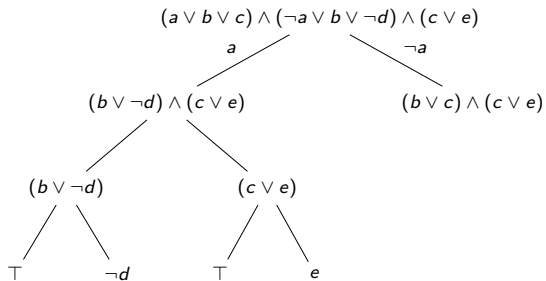
d-DNNF Compilation

DPLL



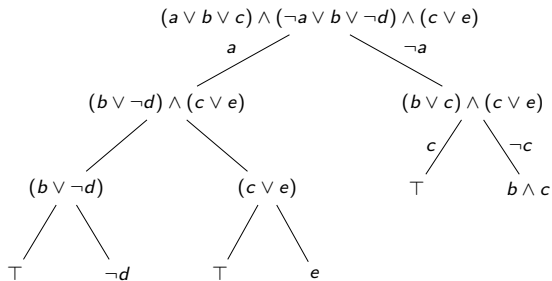
d-DNNF Compilation

DPLL



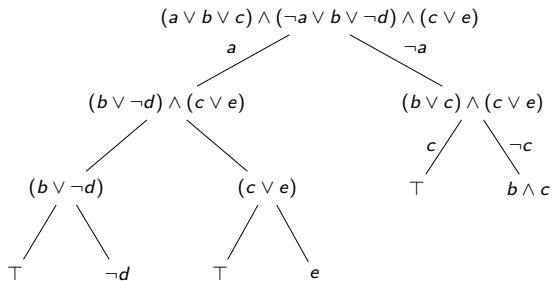
d-DNNF Compilation

DPLL

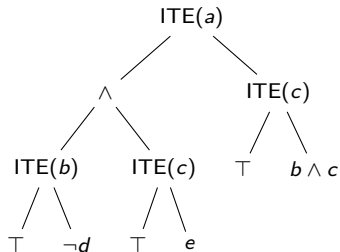


d-DNNF Compilation

DPLL



d-DNNF



ITE = If Then Else

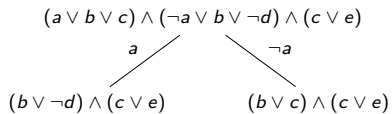
Heuristics

Vanilla DPLL is very slow

Heuristics

Vanilla DPLL is very slow

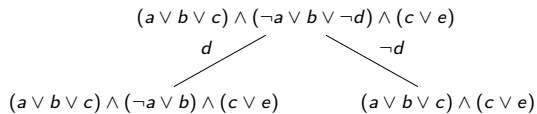
Variable Ordering



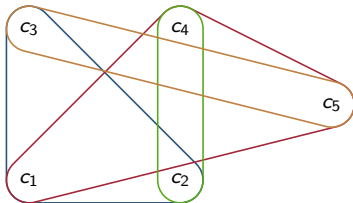
Heuristics

Vanilla DPLL is very slow

Variable Ordering



Heuristics: Dual Hypergraph

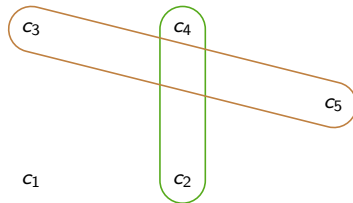
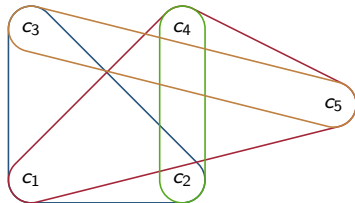


Construction

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

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

Heuristics: Dual Hypergraph



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

$$\begin{array}{c} (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e) \\ \quad \swarrow a \\ (b \vee \neg d) \wedge (c \vee e) \end{array}$$

Projected d-DNNF Compilation

Concept

Projected variables = a, b, d

Sliced variables = c, e

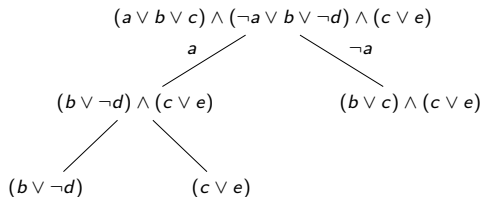
$$\begin{array}{ccc} & (a \vee b \vee c) \wedge (\neg a \vee b \vee \neg d) \wedge (c \vee e) & \\ & \swarrow \quad \searrow & \\ a & & \neg a \\ (b \vee \neg d) \wedge (c \vee e) & & (b \vee c) \wedge (c \vee e) \end{array}$$

Projected d-DNNF Compilation

Concept

Projected variables = a, b, d

Sliced variables = c, e

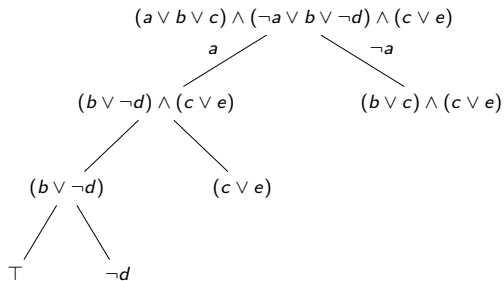


Projected d-DNNF Compilation

Concept

Projected variables = a, b, d

Sliced variables = c, e

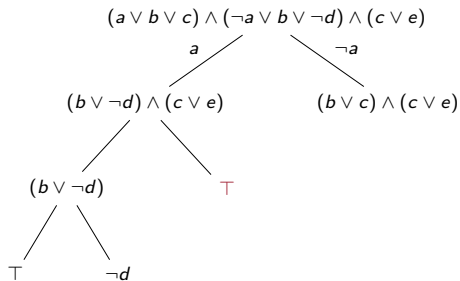


Projected d-DNNF Compilation

Concept

Projected variables = a, b, d

Sliced variables = c, e

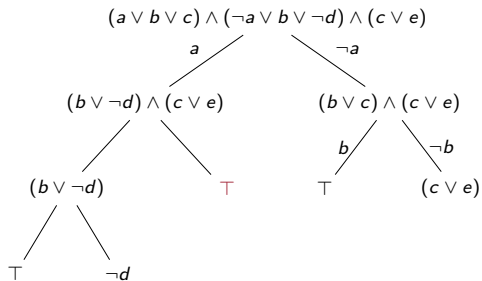


Projected d-DNNF Compilation

Concept

Projected variables = a, b, d

Sliced variables = c, e

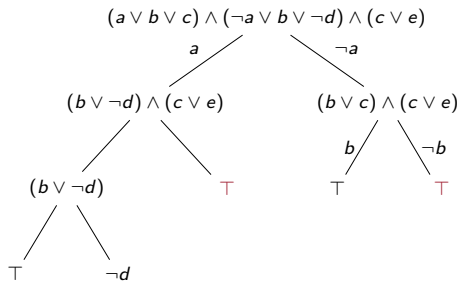


Projected d-DNNF Compilation

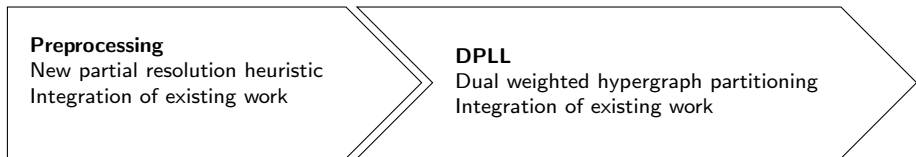
Concept

Projected variables = a, b, d

Sliced variables = c, e

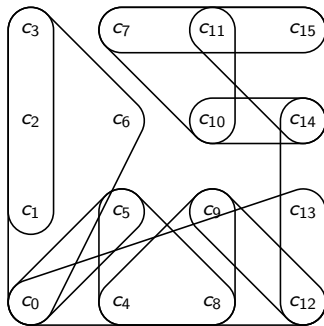


Integration and Optimization in D4²



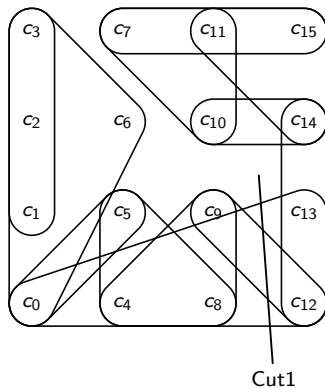
²An Improved Decision-DNNF Compiler, Lagniez et al.

Dual Weighted Hypergraph Partitioning



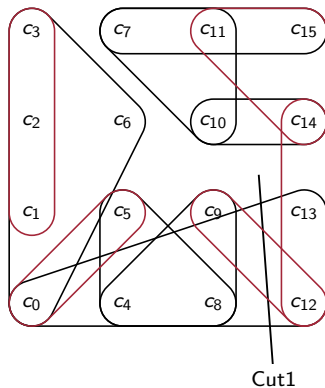
$$F = c_0 \wedge c_1 \wedge \dots \wedge c_{15}$$

Dual Weighted Hypergraph Partitioning



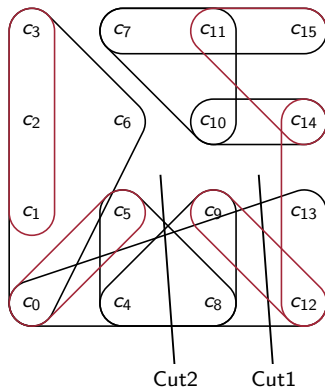
$$F = c_0 \wedge c_1 \wedge \dots \wedge c_{15}$$

Dual Weighted Hypergraph Partitioning



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

Dual Weighted Hypergraph Partitioning



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

Partial Resolution

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

Partial Resolution

Greedy 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

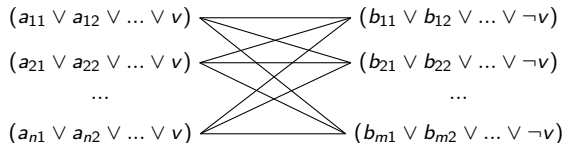
Greedyly resolve "easy" sliced variables until the clause count increases

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

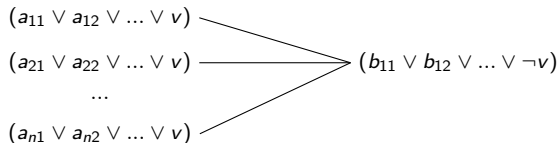
Greedly resolve "easy" sliced variables until the clause count increases

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

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

Connectivity

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

Partial Resolution

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

Connectivity

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

Simpical Variable¹: neighbors form a clique through clauses

¹from GPMC source code

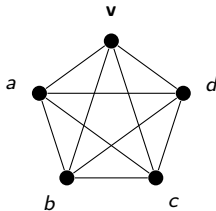
Partial Resolution

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

Connectivity

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

Simpical Variable¹: neighbors form a clique through clauses



¹from GPMC source code

Partial Resolution

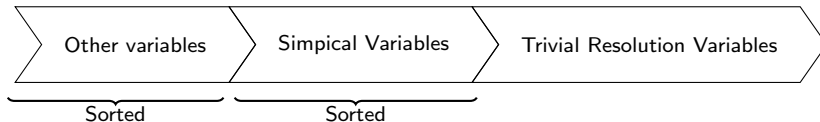
Greedily resolve "easy" sliced variables until the clause count increases

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

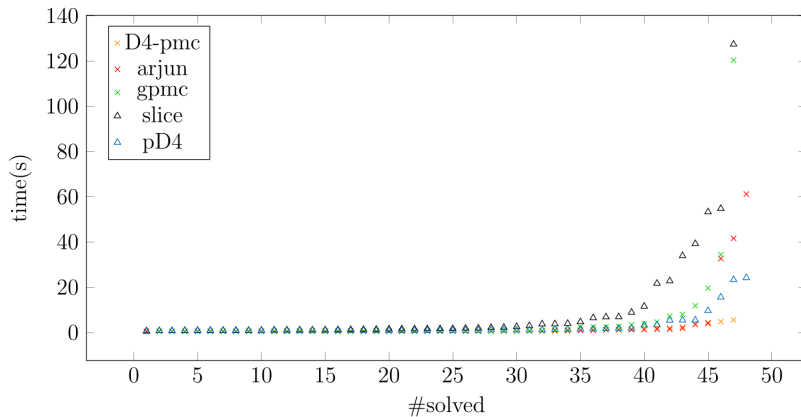
Data

- **Industrial Projection**: Real feature model slicing problems
- **Generated Projection**: Adding randomly selected projected variables to real feature models
- **MC2022**: Private+Public instances from the MC2022 (many unknown sources...)

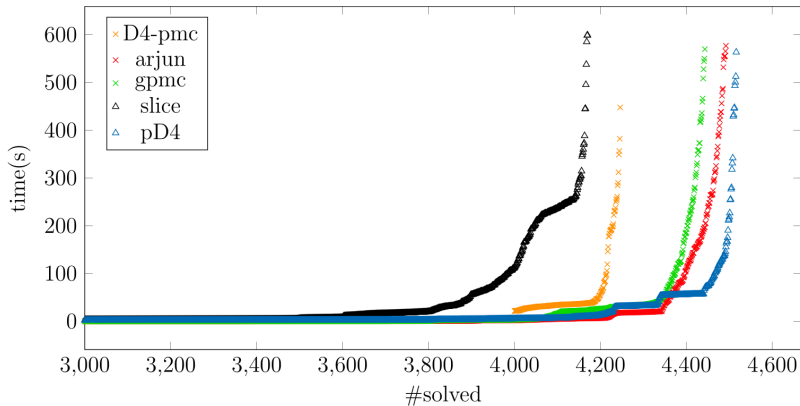
Questions

Compare runtime performance and d-DNNF size

Experiment1: Industrial Projection



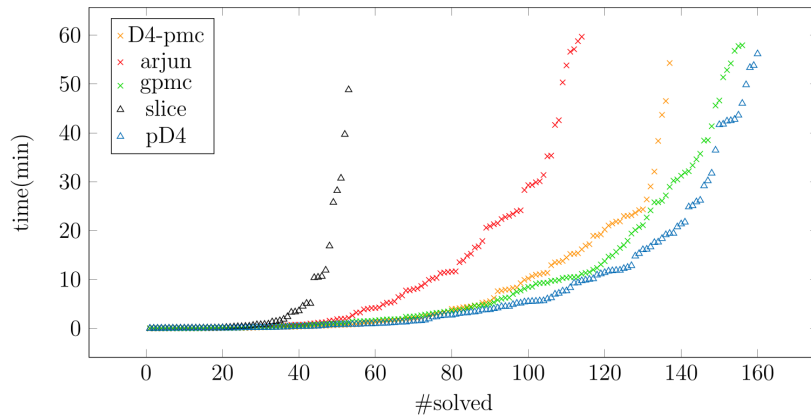
Experiment2: Generated Projection



Experiment2: Generated Projection

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

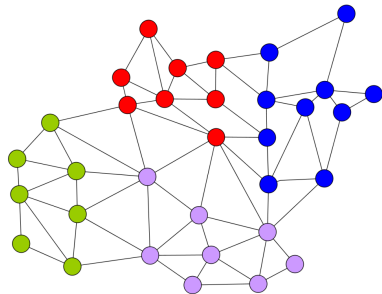
Recap and Feature Work

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

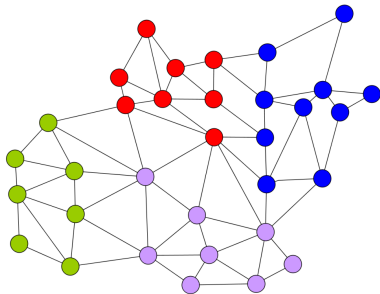
Recap

- 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

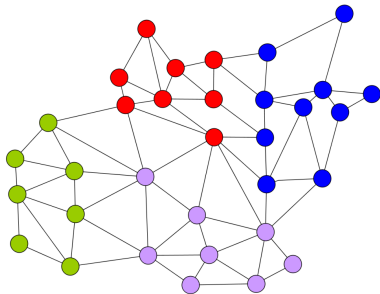
- Combine slice and d-DNNF compilation
- New heuristics for good performance
- Compile hard feature models like linux under projection
- Often comparable or smaller d-DNNF size

Future Work

Lots of new possible heuristics...

Hardware acceleration...

More applications for projected output d-DNNF



The End