Solving (Q)SAT Problems via Tree Decomposition and Dynamic Programming

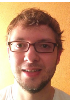
Stefan Woltran

TU Wien, Austria

June 14, 2018

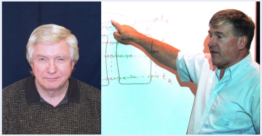
Joint work with

Günther Charwat, Johannes Fichte, Markus Hecher, Michael Morak, Markus Zisser

Introduction

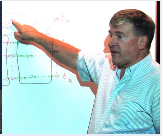
Tree Decomposition and Treewidth



By-product in the theory of graph minors due to Robertson and Seymour (1984); similar notions appeared even earlier (Bertelè and Brioschi, 1972; Halin, 1976).

Introduction

Tree Decomposition and Treewidth

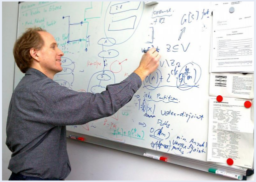
By-product in the theory of graph minors due to Robertson and Seymour (1984); similar notions appeared even earlier (Bertelè and Brioschi, 1972; Halin, 1976).

Courcelle's Theorem (1990)

Any property of finite structures which is definable in MSO can be decided in time $O(f(k) \cdot n)$ where n is the size of the structure and k is its treewidth.

Introduction

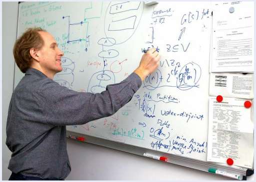
But ...



"*. . . rather than synthesizing methods indirectly from Courcelle's Theorem, one could attempt to develop practical direct methods.*" (Niedermeier, 2006)

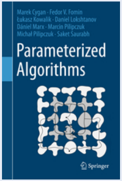
Introduction

But ...



"*...rather than synthesizing methods indirectly from Courcelle's Theorem, one could attempt to develop practical direct methods.*" (Niedermeier, 2006)

... and, more recently ...



"*Courcelle's theorem [...] should be regarded primarily as classification tool, whereas designing efficient dynamic programming routines on tree decompositions requires 'getting your hands dirty' and constructing the algorithm explicitly.* " (Cygan et al., 2015)

Introduction

Main Challenge

Can we turn the huge body of theoretical results on parameterized algorithms into systems that perform competitive in practice?

Introduction

Main Challenge

Can we turn the huge body of theoretical results on parameterized algorithms into systems that perform competitive in practice?

Requirements

1. domain exhibits suitable instances
2. design of smart algorithms and well-engineered systems

Introduction

Main Challenge

Can we turn the huge body of theoretical results on parameterized algorithms into systems that perform competitive in practice?

Requirements

1. domain exhibits suitable instances
2. design of smart algorithms and well-engineered systems

(Most) Competive Arena: SAT problems

- QSAT – propositional logic with quantifiers
- #SAT – model counting
- WMC – weighted model counting
- PMC – projected model counting

Introduction

Main Challenge

Can we turn the huge body of theoretical results on parameterized algorithms into systems that perform competitive in practice?

Requirements

1. domain exhibits suitable instances
2. design of smart algorithms and well-engineered systems

(Most) Competive Arena: SAT problems

- QSAT – propositional logic with quantifiers (PSPACE-complete)
- #SAT – model counting (#P-complete)
- WMC – weighted model counting (#P-complete)
- PMC – projected model counting (#·NP-complete)

Outline

1. Treewidth, Tree Decompositions and Dynamic Programming
2. Solving SAT via TD + DP
3. **dynQBF**: a QSAT solver based on BDDs
4. **gpusat**: a #SAT solver that runs on the GPU
5. Some new results for PMC
6. Conclusion and Outlook

Treewidth and Tree Decompositions

Treewidth

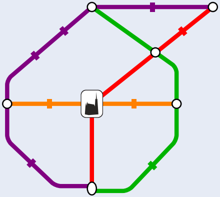
- Some graphs are more “tree-like” than others
- Treewidth measures “tree-likeness”:
  - Trees have treewidth 1
  - The higher the treewidth, the more complex the graph
- Often “easy on trees” implies “easy on tree-like graphs”
  - Many problems are fixed-parameter tractable w.r.t. treewidth k, i.e. can be decided in $O(2^k \cdot n)$
  - That is, they become easy when putting a bound on the treewidth

Treewidth

- Some graphs are more “tree-like” than others
- Treewidth measures “tree-likeness”:
  - Trees have treewidth 1
  - The higher the treewidth, the more complex the graph
- Often “easy on trees” implies “easy on tree-like graphs”
  - Many problems are fixed-parameter tractable w.r.t. treewidth k, i.e. can be decided in $O(2^k \cdot n)$
  - That is, they become easy when putting a bound on the treewidth
- It works for many hard problems
- Real-world applications often have small treewidth

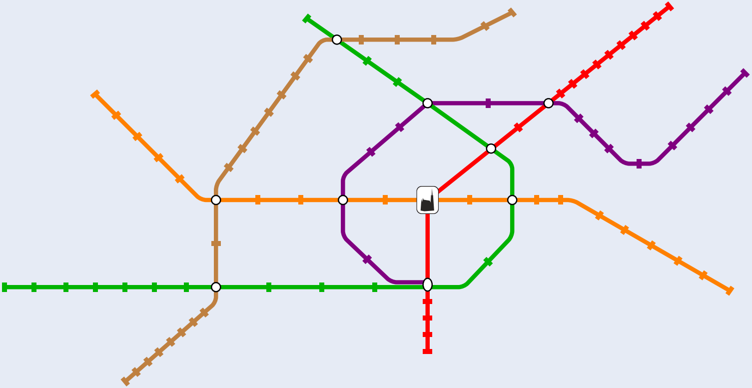
Treewidth (ctd.)

Example: Treewidth 3.



Treewidth (ctd.)

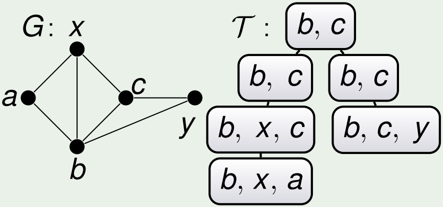
Example: Treewidth 3. Still.



Treewidth is defined in terms of tree decompositions.

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



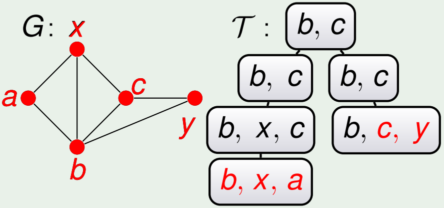
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



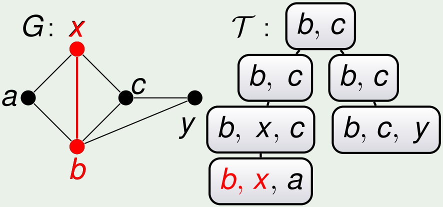
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



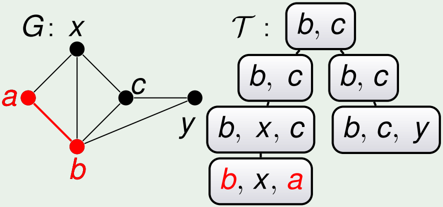
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



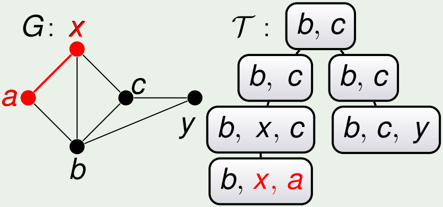
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



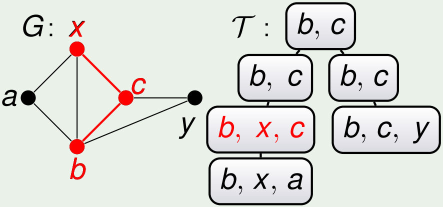
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



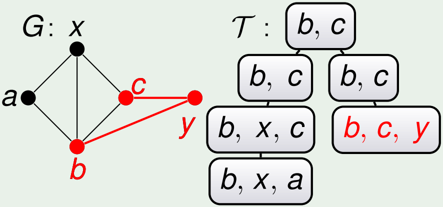
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



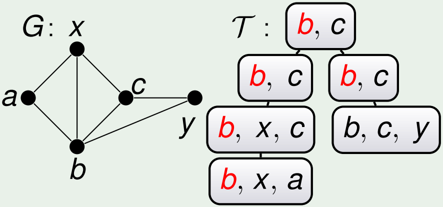
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



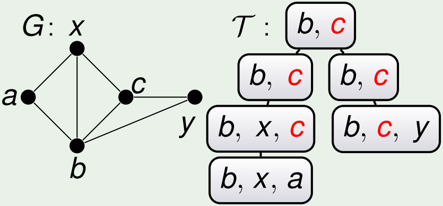
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



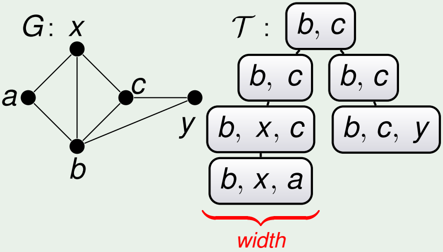
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.

1. each vertex must occur in some *bag*
2. for each edge, there is a bag containing both endpoints
3. tree is *connected*: if v appears in bags of nodes t_0 and t_1, then v is also in the bag of each node on the path between t_0 and t_1

Tree Decompositions

Definition

- width of a decomposition: size of largest bag minus 1
- treewidth of an instance: minimum width over all its TDs

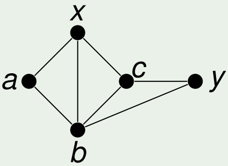
Finding Tree Decompositions

- Constructing a tree decomposition of minimal width intractable
  - but solvable in time $2^{\mathcal{O}(w^3)} \cdot |V|$ [Bodlaender, 1996]
- In Practice:
  - generate a tree decomposition of reasonably low, but not necessarily minimal width using heuristics (e.g. MinFill)
- **htd**: https://github.com/mabseher/htd

Given a tree decomposition of input instance $\mathcal{I}$ of width w, one can solve the problem via dynamic programming in time $f(w) \cdot \mathcal{O}(|\mathcal{I}|^c)$ for some computable function f and constant c.

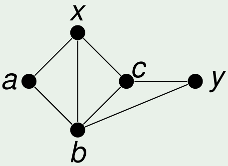
Tree Decompositions

Dynamic Programming - Overall Schema

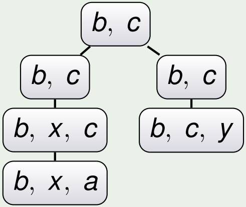


Tree Decompositions

Dynamic Programming - Overall Schema

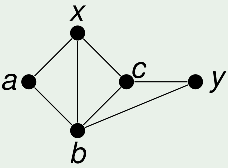


1. Decompose graph

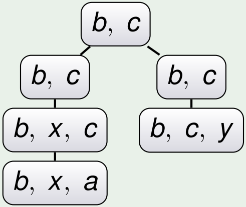


Tree Decompositions

Dynamic Programming - Overall Schema

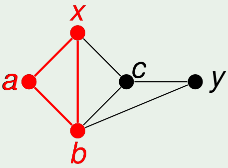


1. Decompose graph
2. Solve subproblems

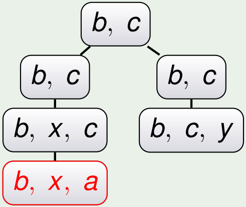


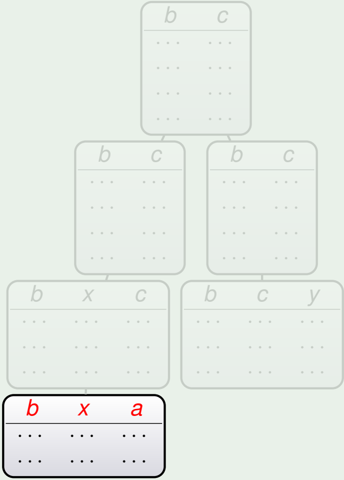
Tree Decompositions

Dynamic Programming - Overall Schema



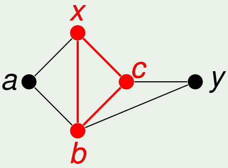
1. Decompose graph
2. Solve subproblems



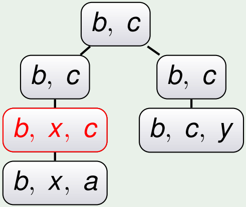


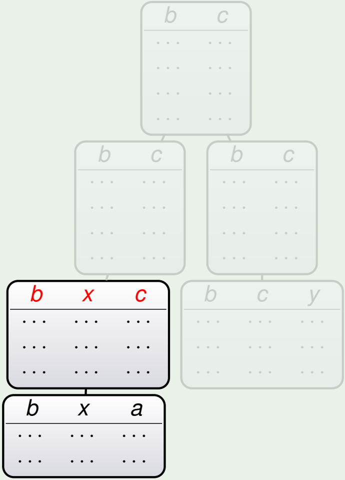
Tree Decompositions

Dynamic Programming - Overall Schema



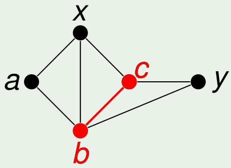
1. Decompose graph
2. Solve subproblems



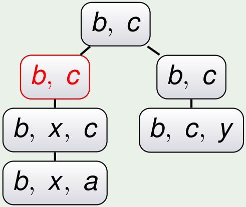


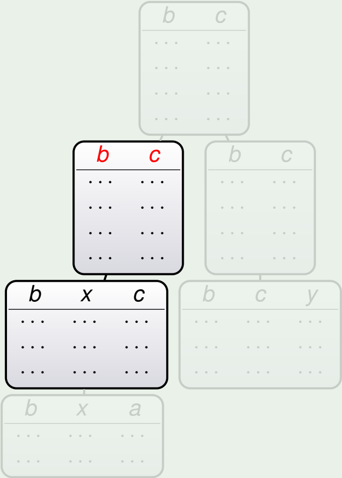
Tree Decompositions

Dynamic Programming - Overall Schema



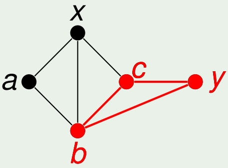
1. Decompose graph
2. Solve subproblems



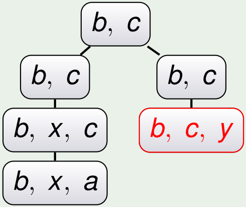


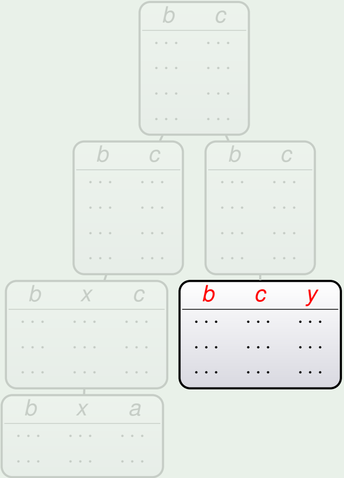
Tree Decompositions

Dynamic Programming - Overall Schema



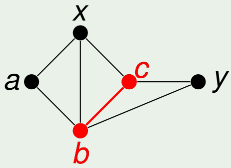
1. Decompose graph
2. Solve subproblems



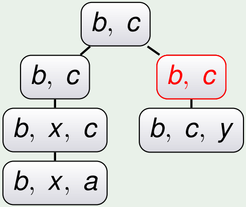


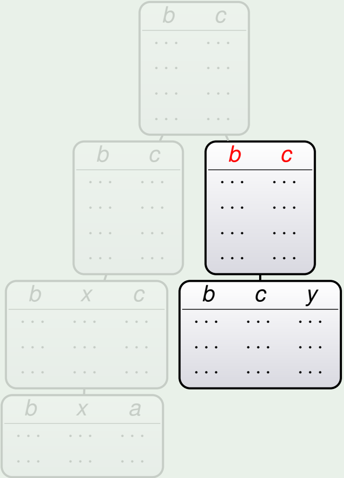
Tree Decompositions

Dynamic Programming - Overall Schema



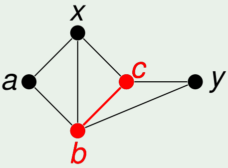
1. Decompose graph
2. Solve subproblems



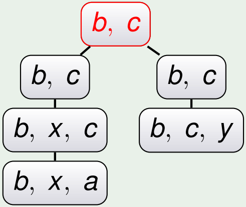


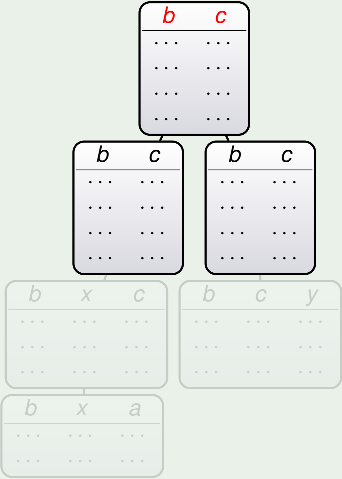
Tree Decompositions

Dynamic Programming - Overall Schema



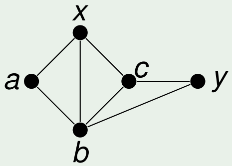
1. Decompose graph
2. Solve subproblems



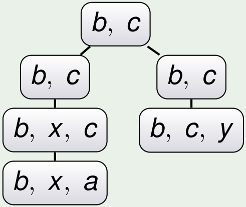


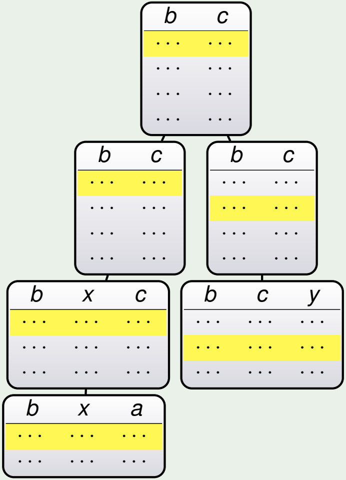
Tree Decompositions

Dynamic Programming - Overall Schema



1. Decompose graph
2. Solve subproblems
3. Output solutions





Let’s Get Things Started: Solving SAT via TD and DP

DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

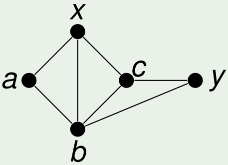
DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

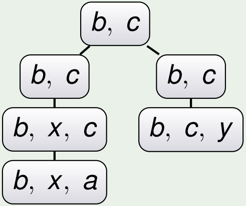
$$\begin{aligned} Mod(\varphi) = \{ \quad & \{b\}, \{a,b\}, \{b,c\}, \{a,b,c\}, \\ & \{b,c,x\}, \{a,b,c,x\}, \\ & \{b,y\}, \{a,b,y\}\} \end{aligned}$$

DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

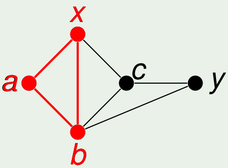


1. Decompose graph

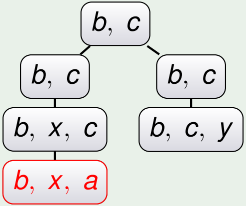


DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



b	c
1	0
1	1

b	c
1	0
1	1

b	c
0	0
1	0
1	1

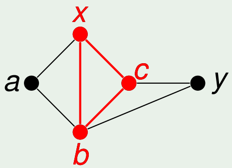
b	x	c
1	0	0
1	0	1
1	1	1

b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

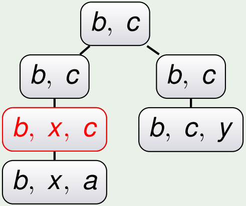
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



b	c
1	0
1	1

b	c
1	0
1	1

b	c
0	0
1	0
1	1

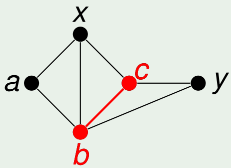
b	x	c
1	0	0
1	0	1
1	1	1

b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

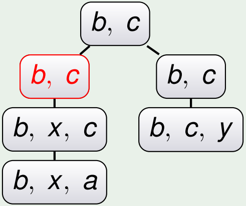
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



b	c
1	0
1	1

b	c
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

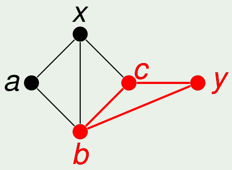
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

b	c
0	0
1	0
1	1

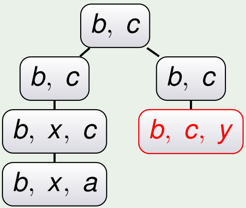
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



b	c
1	0
1	1

b	c
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

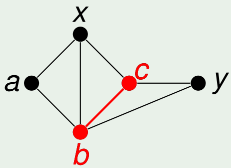
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

b	c
0	0
1	0
1	1

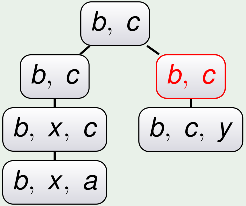
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



b	c
1	0
1	1

b	c
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

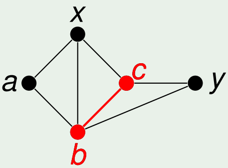
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

b	c
0	0
1	0
1	1

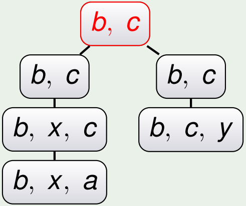
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

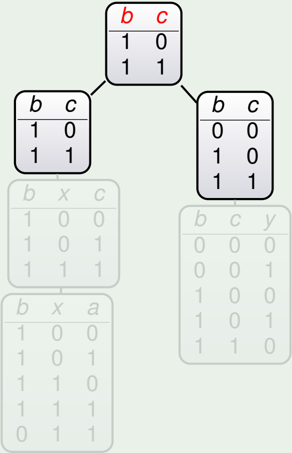
DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



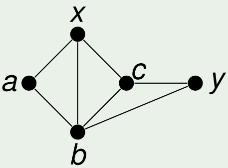
1. Decompose graph
2. Solve subproblems



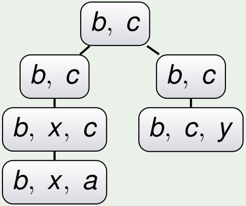


DP algorithm for SAT [Samer & Szeider, 2010]

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems
3. Counting solutions



b	c	#
1	0	4
1	1	4

b	c	#
1	0	2
1	1	4

b	c	#
0	0	2
1	0	2
1	1	1

b	x	c	#
1	0	0	2
1	0	1	2
1	1	1	2

b	c	y	#
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1
1	1	0	1

b	x	a	#
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1
0	1	1	1

Put Things on Track: QSAT via TD and DP

QSAT

- Extension of propositional logic
- Compactly encode computationally hard problems (e.g., verification, planning, synthesis, ...)
- Satisfiability problem (QSAT) is PSPACE-complete
- Various techniques: search (DPLL, CDCL), expansion, resolution, CEGAR
- Annual QBF Competition (47 systems submitted in 2017)

Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

Recall,

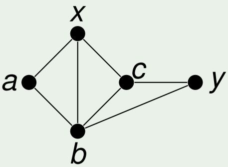
$$\begin{aligned} Mod(\varphi) \ = \{ \quad & \{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \\ & \{b, c, x\}, \{a, b, c, x\}, \\ & \{b, y\}, \{a, b, y\}\} \end{aligned}$$

Hence, Φ invalid:

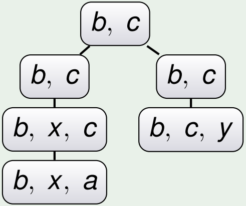
$$\varphi[x = 1, y = 1] \equiv (a \vee b) \wedge c \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c) \equiv \bot$$

Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

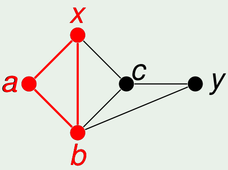


1. Decompose graph

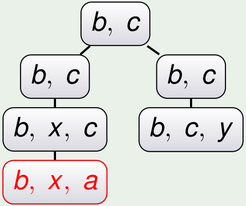


Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where
$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$



1. Decompose graph
2. Solve subproblems



$(\neg x, \neg y)$

b	c
1	0
1	1

$(\neg x, y)$

b	c
1	0

$(x, \neg y)$

b	c
1	1

(x, y)

b	c

$(\neg x)$

b	c
1	0
1	1

(x)

b	c
1	1

$(\neg y)$

b	c
0	0
1	0
1	1

(y)

b	c
0	0
1	0

b	x	c
1	0	0
1	0	1
1	1	1

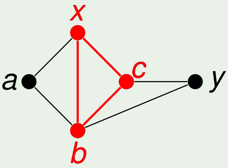
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

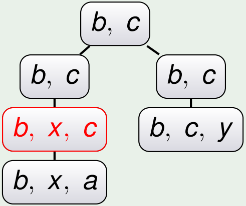
Dynamic Programming for QSAT

$\Phi = \forall x, y \; \exists a, b, c \; \varphi$, where

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



$(\neg x, \neg y)$

b	c
1	0
1	1

$(\neg x, y)$

b	c
1	0

$(x, \neg y)$

b	c
1	1

(x, y)

b	c

$(\neg x)$

b	c
1	0
1	1

(x)

b	c
1	1

$(\neg y)$

b	c
0	0
1	0
1	1

(y)

b	c
0	0
1	0

b	x	c
1	0	0
1	0	1
1	1	1

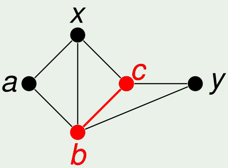
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

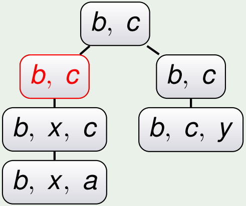
Dynamic Programming for QSAT

$\Phi = \forall x, y \, \exists a, b, c \; \varphi$, where

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



$(\neg x, \neg y)$

b	c
1	0
1	1

$(\neg x, y)$

b	c
1	0

$(x, \neg y)$

b	c
1	1

(x, y)

b	c

$(\neg x)$

b	c
1	0
1	1

(x)

b	c
1	1

b	x	c
1	0	0
1	0	1
1	1	1

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

$(\neg y)$

b	c
0	0
1	0
1	1

(y)

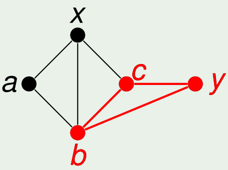
b	c
0	0
1	0

b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

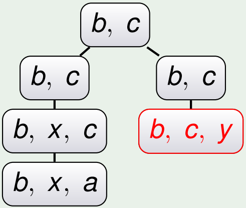
Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



$(\neg x, \neg y)$

b	c
1	0
1	1

$(\neg x, y)$

b	c
1	0

$(x, \neg y)$

b	c
1	1

(x, y)

b	c

$(\neg x)$

b	c
1	0
1	1

(x)

b	c
1	1

$(\neg y)$

b	c
0	0
1	0
1	1

(y)

b	c
0	0
1	0

b	x	c
1	0	0
1	0	1
1	1	1

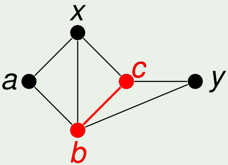
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

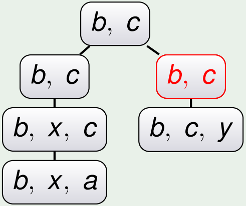
Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



$(\neg x, \neg y)$

b	c
1	0
1	1

$(\neg x, y)$

b	c
1	0

$(x, \neg y)$

b	c
1	1

(x, y)

b	c

$(\neg x)$

b	c
1	0
1	1

(x)

b	c
1	1

b	x	c
1	0	0
1	0	1
1	1	1

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

$(\neg y)$

b	c
0	0
1	0
1	1

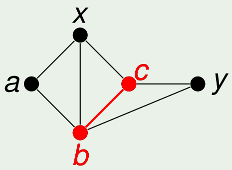
(y)

b	c
0	0
1	0

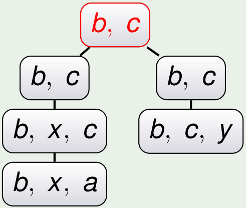
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems



$(\neg x, \neg y)$

b	c
1	0
1	1

$(\neg x, y)$

b	c
1	0

$(x, \neg y)$

b	c
1	1

(x, y)

b	c

$(\neg x)$

b	c
1	0
1	1

(x)

b	c
1	1

$(\neg y)$

b	c
0	0
1	0
1	1

(y)

b	c
0	0
1	0

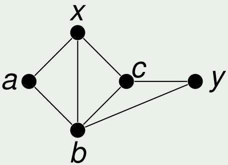
b	x	c
1	0	0
1	0	1
1	1	1

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

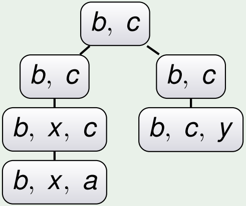
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems
3. Evaluation of root



$(\neg x, \neg y)$

b	c
1	0
1	1

$(\neg x, y)$

b	c
1	0

$(x, \neg y)$

b	c
1	1

(x, y)

b	c

$(\neg x)$

b	c
1	0
1	1

(x)

b	c
1	1

$(\neg y)$

b	c
0	0
1	0
1	1

(y)

b	c
0	0
1	0

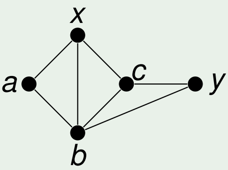
b	x	c
1	0	0
1	0	1
1	1	1

b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

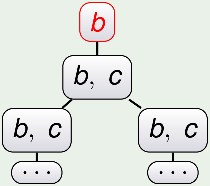
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

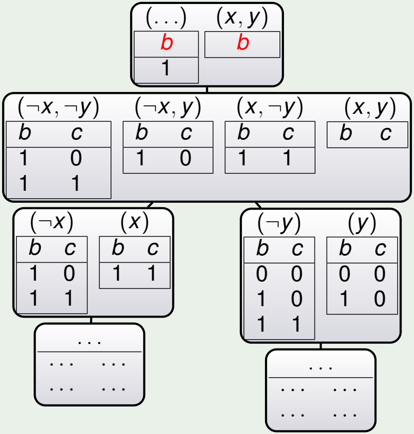
Dynamic Programming for QSAT

$\Phi = \forall x, y\ \exists a, b, c\ \varphi$, where
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems
3. Evaluation of root

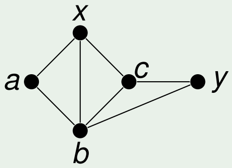




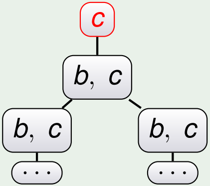
Dynamic Programming for QSAT

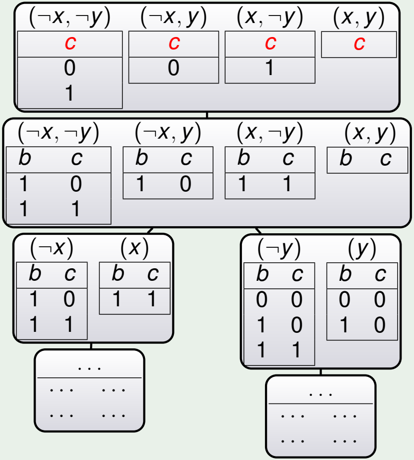
$\Phi = \forall x, y \; \exists a, b, c \; \varphi$, where

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems
3. Evaluation of root





Data Structure

Binary Decision Diagrams (BDDs)

- BDDs store Boolean functions as rooted DAG
- Reduced Ordered BDDs
  - Usually space-efficient (given a good variable ordering)
  - Canonical (equivalent formulae represented by same BDD)

Data Structure

Binary Decision Diagrams (BDDs)

- BDDs store Boolean functions as rooted DAG
- Reduced Ordered BDDs
  - Usually space-efficient (given a good variable ordering)
  - Canonical (equivalent formulae represented by same BDD)

Nested Set of Formulae (NSF)

- Innermost elements are BDDs and store parts of the QBF matrix
- Nestings of depth k account for quantifier blocks in the prefix

Data Structure

Binary Decision Diagrams (BDDs)

- BDDs store Boolean functions as rooted DAG
- Reduced Ordered BDDs
  - Usually space-efficient (given a good variable ordering)
  - Canonical (equivalent formulae represented by same BDD)

Nested Set of Formulae (NSF)

- Innermost elements are BDDs and store parts of the QBF matrix
- Nestings of depth k account for quantifier blocks in the prefix

Example

NSF: $\{\{\{\top, \bot\}\}, \{\{\neg a \vee b\}, \{\bot\}, \{a \wedge b\}\}\}$ with $k = 3$

Data Structure

Binary Decision Diagrams (BDDs)

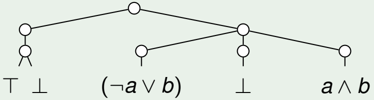
- BDDs store Boolean functions as rooted DAG
- Reduced Ordered BDDs
  - Usually space-efficient (given a good variable ordering)
  - Canonical (equivalent formulae represented by same BDD)

Nested Set of Formulae (NSF)

- Innermost elements are BDDs and store parts of the QBF matrix
- Nestings of depth k account for quantifier blocks in the prefix

Example

NSF: $\{\{\{\top, \bot\}\}, \{\{\neg a \vee b\}, \{\bot\}, \{a \wedge b\}\}\}$ with $k = 3$



Run Time

Given QBF $Q_1 X_1 \ldots Q_k X_k \psi$ and a tree decomposition for ψ of width w.

The algorithm determines the truth value of the QBF in time

$$\mathcal{O}(2^{2^{\cdot^{\cdot^{2^{w+1}}}}} \cdot |\psi|),$$

where the height of the tower of exponents is $k + 1$:

- the size of each BDD is at most 2^{w+1}
- k quantifier blocks

Run Time

Given QBF $Q_1 X_1 \ldots Q_k X_k \psi$ and a tree decomposition for ψ of width w.

The algorithm determines the truth value of the QBF in time

$$\mathcal{O}(2^{2^{\cdot^{\cdot^{2^{w+1}}}}} \cdot |\psi|),$$

where the height of the tower of exponents is $k + 1$:

- the size of each BDD is at most 2^{w+1}
- k quantifier blocks

- QSAT is fixed-parameter tractable for bounded treewidth and the number of quantifier alternations [Chen, 2004]
- QSAT is *not* fixed-parameter tractable w.r.t. parameter treewidth only [Atserias and Oliva, 2014].

Towards Efficiency in Practice

Clause splitting

- Size of largest clause gives lower bound for width
- Splitting usually reduces the width (but increases the number of variables)

Dependency Schemes

- A *dependency scheme D* is an overapproximation of full independence. (independence: reordering of quantifiers does not change satisfiability)
- If a variable is removed, we only need to split tables if a dependent variable is not yet fully processed

Feature-based tree decomposition selection

- Choose tree decomposition based on certain criteria (besides width)
- Promising: variable position, children of join nodes

Towards Efficiency in Practice

Intermediate unsatisfiability checks

- Reuse procedure for deciding the problem on NSFs obtained during bottom-up traversal
- If procedure returns $\perp$, the instance is unsatisfiable
- For $\top$, the QBF might still be unsatisfiable due to clauses that were not yet considered

Subset-based compression

- Check for subsets w.r.t. models represented by the BDDs and subsets w.r.t. nested sets
- Similar to subsumption checking [Biere, 2004]

Balance NSF and BDD size

- Delay splitting of removed variables (store them in a cache)
- Increases size of BDDs (no longer bounded by width)
- Apply heuristics to obtain optimal NSF and BDD size

The dynQBF System

System Specifics

- C++, open source
- Tree decomposition: **htd** library
- BDD management: CUDD
- Standard dependencies (optional): DepQBF

Core Features

- Deciding QSAT
- Partial certificates (outermost quantifier block)
  - Compact enumeration
  - Counting

https://github.com/gcharwat/dynqbf

Experiments: Setup

QBF solvers that participated in the 2016 QBF Evaluation

- Top-ranked in the competition
- Publicly available
- Without (explicit) tool-chained preprocessing

Experiments: Setup

QBF solvers that participated in the 2016 QBF Evaluation

- Top-ranked in the competition
- Publicly available
- Without (explicit) tool-chained preprocessing

2016 QBF Evaluation instances (preprocessed with Bloqqer)

- 2-QBF track: 305 instances, 130 solved by Bloqqer
- PCNF track: 825 instances, 341 solved by Bloqqer

Experiments: Setup

QBF solvers that participated in the 2016 QBF Evaluation

- Top-ranked in the competition
- Publicly available
- Without (explicit) tool-chained preprocessing

2016 QBF Evaluation instances (preprocessed with Bloqqer)

- 2-QBF track: 305 instances, 130 solved by Bloqqer
- PCNF track: 825 instances, 341 solved by Bloqqer

Run limitations and measurements

- Ranked by number of solved instances
- Timeout: 10 minutes; Memout: 16 GB
- Given solving time (in seconds) includes penalty of 10 minutes for non-solved instances
- Detailed analysis w.r.t. width of instances

Experiments: 2-QBF instances

System	Solved	Time	SAT	UNSAT	Unique
Qesto	236	50K	160	76	0
RAReQS	232	51K	161	71	1
dynQBF	221	53K	172	49	43
DepQBF	221	56K	143	78	1
QSTS	220	58K	162	58	2
CAQE	204	65K	153	51	0
AReQS	202	66K	141	61	0
GhostQ (CEGAR)	151	95K	123	28	0

Table: Data set: QBFEval'16 – 2-QBF track, preprocessed with Bloqqer

Uniquely solved instances

dynQBF: fixpoint detection (43)
QSTS: query (1), sorting networks (1)

Experiments: 2-QBF instances

$w \leq 80$ (86 instances)			$w > 80$ (89 instances)		
System	Solved	Time	System	Solved	Time
dynQBF	79	6K	RAReQS	69	17K
DepQBF	41	28K	QSTS	69	18K
Qesto	39	31K	Qesto	67	19K
RAReQS	33	34K	DepQBF	50	28K
CAQE	28	36K	AReQS	47	28K
AReQS	25	38K	CAQE	46	29K
QSTS	21	40K	**dynQBF**	12	47K
GhostQ (CEGAR)	9	47K	GhostQ (CEGAR)	12	49K

Table: Data set: QBFEval'16 – 2-QBF track, 175 non-trivial instances after preprocessing

Experiments: PCNF instances

System	Solved	Time	SAT	UNSAT	Unique
RAReQS	633	126K	301	332	14
Qesto	618	134K	298	320	1
DepQBF	596	144K	296	300	7
QSTS	592	149K	294	298	3
CAQE	589	155K	295	294	1
GhostQ (CEGAR)	571	161K	293	278	1
dynQBF	494	203K	239	255	21

Table: Data set: QBFEval'16 – PCNF track, preprocessed with Bloqqer

Uniquely solved instances

dynQBF: fixpoint detection (11), ...
RAReQS: dungeon/planning (3), emptyroom (3), ...

Experiments: PCNF instances

$w \leq 80$ (182 instances)			$w > 80$ (302 instances)		
System	Solved	Time	System	Solved	Time
RAReQS	137	28K	RAReQS	155	98K
dynQBF	134	32K	Qesto	148	100K
Qesto	129	34K	DepQBF	131	108K
DepQBF	124	36K	CAQE	129	114K
QSTS	123	37K	QSTS	128	112K
CAQE	119	40K	GhostQ (CEGAR)	112	120K
GhostQ (CEGAR)	118	41K	**dynQBF**	19	171K

Table: Data set: QBFEval'16 – PCNF track, 484 non-trivial instances after preprocessing

QSAT solving — Summary

A novel expansion-based approach for QBF solving

- Motivated by fixed-parameter tractability results
- Explicitly takes QBF structure into account
- Various optimizations towards feasibility in practice

QSAT solving — Summary

A novel expansion-based approach for QBF solving

- Motivated by fixed-parameter tractability results
- Explicitly takes QBF structure into account
- Various optimizations towards feasibility in practice

QBF solver dynQBF

- Competitive on instances up to treewidth 80, and 2-QBF instances
- Many uniquely solved instances
- 2-QBF track: Ranked 8 (out of 29 participants) in QBFEval'17
- PCNF track: Ranked 13 (out of 30 participants) in QBFEval'17

Massive Parallelisation: #SAT on the GPU

#SAT

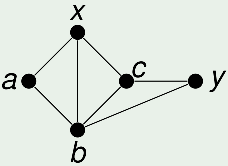
- Given propositional formula φ, #SAT asks for the number of models

$$|\{M \subseteq var(\varphi) \mid M \models \varphi\}|$$

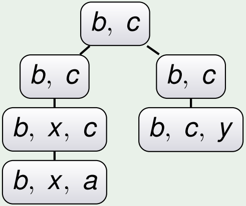
- Applications in several domains, e.g.:
  - Bayesian reasoning [Sang et al., 05]
  - Infrastructure reliability [Meel et al., 17]
- Traversal of entire search space required
- Systems relying on different approaches exist
  - Cachet, sharpSAT;
  - ApproxMC, sts;
  - countAntom;
  - c2d, d4;
  - ...

Recall: DP for #SAT

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. Decompose graph
2. Solve subproblems
3. Counting solutions



b	c	y	#
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1
1	1	0	1

b	c	#
1	0	2
1	1	4

b	x	c	#
1	0	0	2
1	0	1	2
1	1	1	2

b	x	a	#
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1
0	1	1	1

b	c	#
0	0	2
1	0	2
1	1	1

b	c	y	#
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1
1	1	0	1

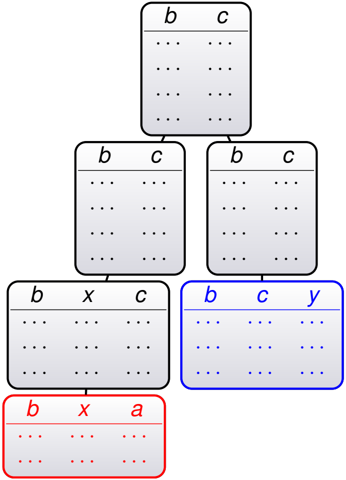
DP on the GPU

How to parallelize DP?

DP on the GPU

How to parallelize DP?

1. Compute tables in parallel
   - No massive parallelization due to dependencies of child nodes

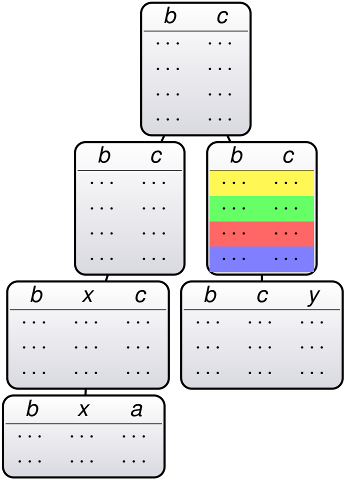


DP on the GPU

How to parallelize DP?

1. Compute tables in parallel
   - No massive parallelization due to dependencies of child nodes
2. Compute rows in parallel
   - Since computation of specific rows is independet of other rows, this allows for massive parallelization

$\Rightarrow$ Used here!



The gpuSAT System

System Specifics

- C++11
- Tree decomposition: **htd** library
- OpenCL
    - Vendor independent
    - C99
    - SIMT

Core Features

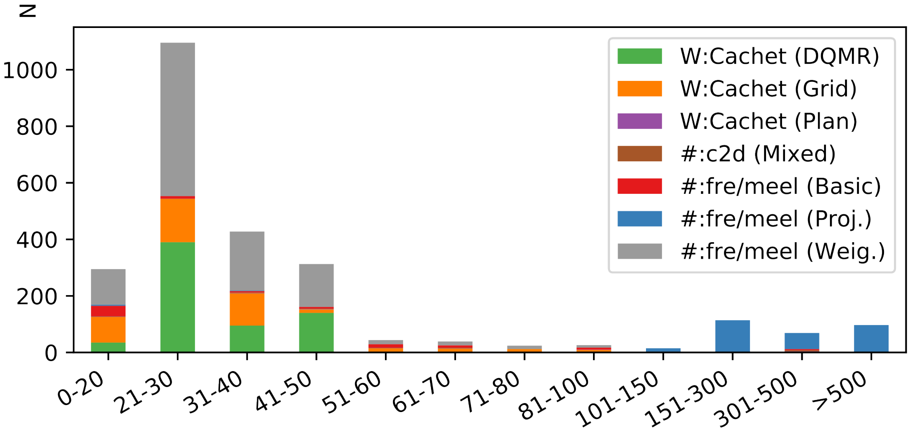
- Each potential row of a table runs in one thread of the GPU
- Tables need to be split for $w > 27$
- Precision can be toggled

https://github.com/Budddy/GPUSAT

Experiments

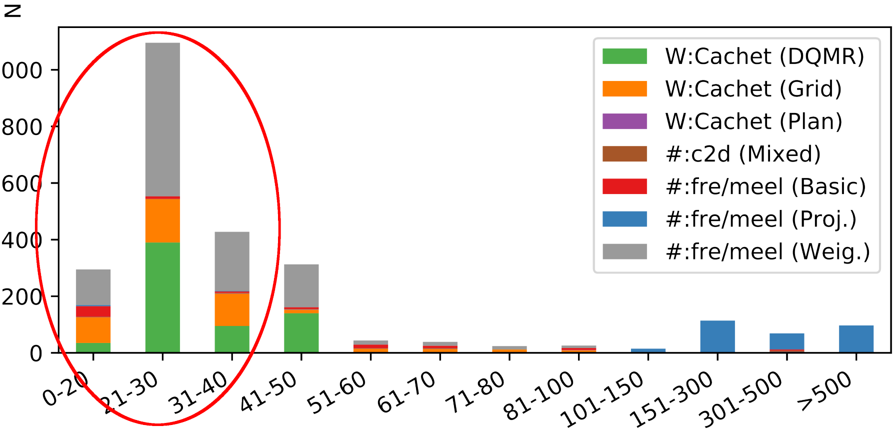
- Timeout: 900s
- Memory: 8GB
- 16 #SAT Solvers
- 3 WMC Solvers
- 5 random TDs per instance

Width Distribution of Instances



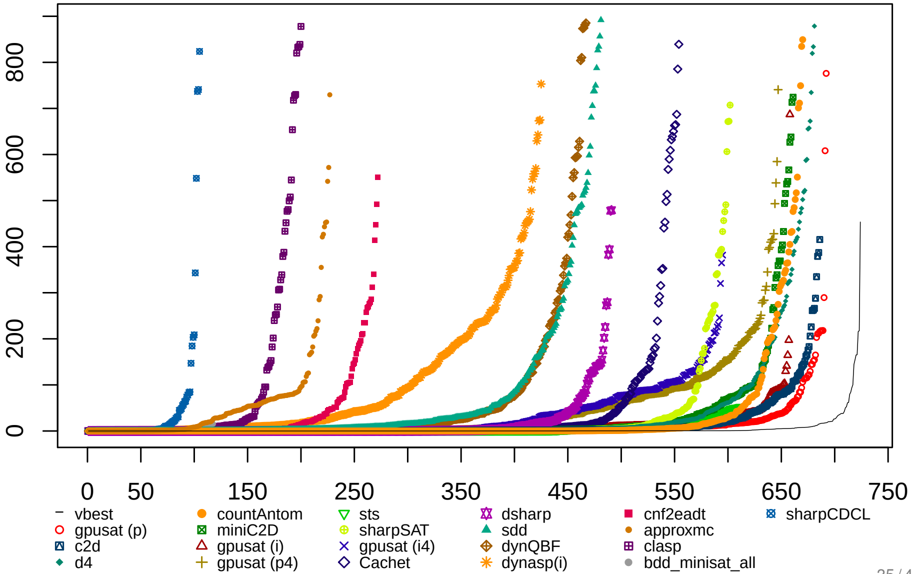
- Instances: 1091 WMC and 1456 #SAT

Width Distribution of Instances



- Instances: 1091 WMC and 1456 #SAT
- 54% $\leq$ width 30;   70% $\leq$ width 40
- Decomposition time below a second (max 2.5s)

Results #SAT (Width: 0-30)



#SAT solving — Summary

Towards DP on the GPU

- Distribute computation of tables among different computation units
- All threads have same instructions but start from different data
- Each row forms a potential pixel with the corresponding sum as its value

Prototype System **gpuSAT**

- Competitive up to width 30; solved instances up to width 45
- High Precision
- Easily extendible to WMC (also supported by gpuSAT)

Side Result: DP for PMC

PMC

- Given propositional formula φ and $P \subseteq \mathit{var}(\varphi)$, $\mathrm{PMC}_P(\varphi)$ asks for the number of P-projected models

$$|\{M \cap P \mid M \subseteq \mathit{var}(\varphi), M \models \varphi\}|$$

- Extremal Cases
  - $P = \emptyset$ amounts to SAT
  - $P = \mathit{var}(\varphi)$ amounts to #SAT

- However, the problem is in general harder than #SAT ($\#\cdot$NP-complete vs. #P-complete)

Towards Dynamic Programming for PMC

Theorem

Unless ETH fails, there is no algorithm for PMC running in time $2^{2^{o(tw)}} \cdot |\varphi|^c$.

Towards Dynamic Programming for PMC

Theorem

Unless ETH fails, there is no algorithm for PMC running in time $2^{2^{o(tw)}} \cdot |\varphi|^c$.

Proof.

- Unless ETH fails, 2-QSAT can not be solved [Lampis & Mitsou, 2017] in time $2^{2^{o(tw)}} \cdot |\varphi|^c$.

Towards Dynamic Programming for PMC

Theorem

Unless ETH fails, there is no algorithm for PMC running in time $2^{2^{o(tw)}} \cdot |\varphi|^c$.

Proof.

- Unless ETH fails, 2-QSAT can not be solved [Lampis & Mitsou, 2017] in time $2^{2^{o(tw)}} \cdot |\varphi|^c$.
- Solve $\forall X.\exists Y.\varphi$ by checking whether $\mathrm{PMC}_X(\varphi) = 2^{|X|}$.

$\square$

Dynamic Programming for PMC

Can we find a DP that (asymptotically) matches this lower bound.

Dynamic Programming for PMC

Can we find a DP that (asymptotically) matches this lower bound.

Yes ...

PMC in three Steps

1. Run algorithm for SAT

Dynamic Programming for PMC

Can we find a DP that (asymptotically) matches this lower bound.

Yes ...

PMC in three Steps

1. Run algorithm for SAT
2. Purge non-solutions

Dynamic Programming for PMC

Can we find a DP that (asymptotically) matches this lower bound.

Yes ...

PMC in three Steps

1. Run algorithm for SAT
2. Purge non-solutions
3. Add projection counters for sets of rows

Dynamic Programming for PMC

$P = \{x, y\}$
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

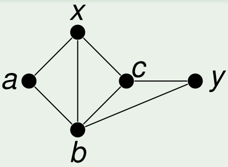
$$\begin{aligned} Mod(\varphi) \quad = \{ \quad & \{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \\ & \{b, c, x\}, \{a, b, c, x\}, \\ & \{b, y\}, \{a, b, y\}\} \end{aligned}$$

$\mathrm{PMod}_P(\varphi) = \{\emptyset, \{x\}, \{y\}\}$
$\mathrm{PMC}_P(\varphi) = |\mathrm{PMod}_P(\varphi)| = 3$

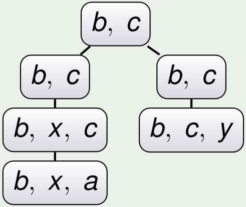
Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. DP for SAT



b	c
1	0
1	1

b	c
1	0
1	1

b	x	c
1	0	0
1	0	1
1	1	1

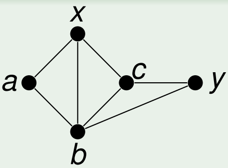
b	x	a
1	0	0
1	0	1
1	1	0
0	1	1
1	1	1

b	c
0	0
1	0
1	1

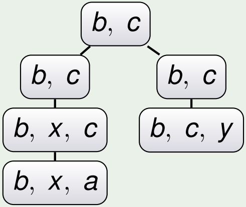
b	c	y
0	0	0
1	0	0
1	1	0
0	0	1
1	0	1

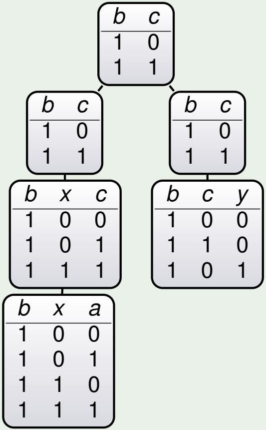
Dynamic Programming for PMC

$P = \{x, y\}$
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$



1. DP for SAT
2. Purge non-solutions





Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

b	c	ipmc	
1	0	2	1
1	1	2	

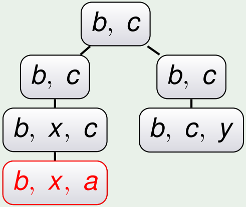
b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



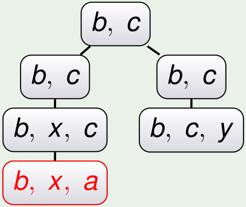
b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

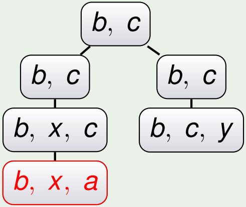
b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

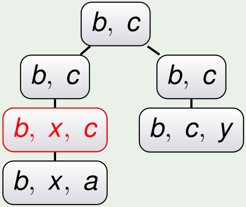
b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

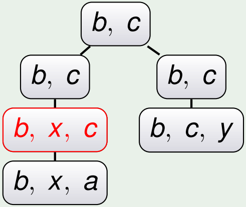
b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

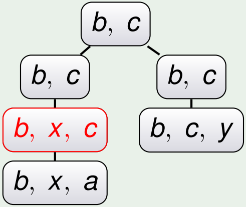
b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

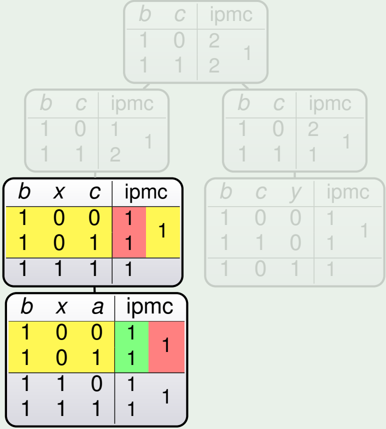
b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



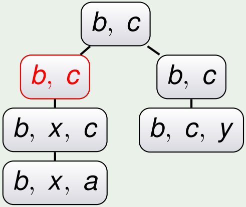


Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$
$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

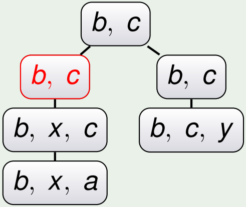
b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

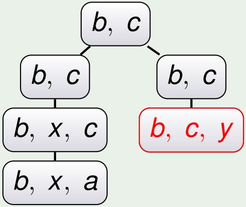


Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

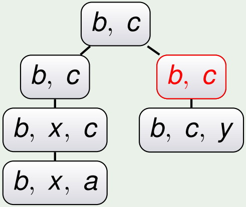
b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

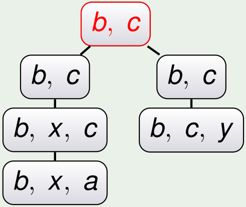
b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

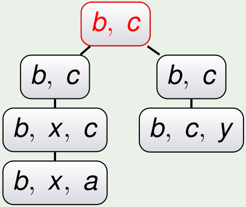
b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

Dynamic Programming for PMC

$P = \{x, y\}$

$\varphi = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

1. DP for SAT
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$



b	c	ipmc	
1	0	2	1
1	1	2	

b	c	ipmc	
1	0	1	1
1	1	2	

b	c	ipmc	
1	0	2	1
1	1	1	

b	x	c	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	1	1	

b	c	y	ipmc	
1	0	0	1	1
1	1	0	1	
1	0	1	1	

b	x	a	ipmc	
1	0	0	1	1
1	0	1	1	
1	1	0	1	1
1	1	1	1	

PMC – Summary

Contribution

- Algorithm for PMC using treewidth with worst-case runtime $2^{2^{\mathcal{O}(tw)}} \cdot |\varphi| \cdot \gamma(|\varphi|)$
  - Relies on inclusion/exclusion principle
  - In the worst-case we require 2^k counters for k rows in a table
- Unless ETH fails, there is no algorithm for PMC running in time $2^{2^{o(tw)}} \cdot |\varphi|^c$.

Conclusion and Outlook

Conclusion and Outlook

Lessons learned:

- DP on TD efficient in practice (at least if width not too high)
- However, engineering efforts required
  - smart data-structures help a lot (BDDs)
  - (simple) DP allows for massive parallelisation
  - shape of TD crucial

Conclusion and Outlook

Lessons learned:

- DP on TD efficient in practice (at least if width not too high)
- However, engineering efforts required
  - smart data-structures help a lot (BDDs)
  - (simple) DP allows for massive parallelisation
  - shape of TD crucial

How to tame high treewidth / performance bottlenecks?

- width-reducing preprocessing
- abstraction / hybrid solving
- relaxed decompositions [Maniu, Senellart, Jog; 2017]
- other type of TD heuristics needed [Jégou, Kanso, Terrioux; 2016]
- lazy materialization of tables in DP

References

- Günther Charwat, Stefan Woltran: Expansion-based QBF Solving on Tree Decompositions. RCRA@AI*IA 2017: 16-26
- Johannes Fichte, Markus Hecher, Michael Morak and Stefan Woltran: Exploiting Treewidth for Projected Model Counting and its Limits. SAT 2018. To appear.
- Johannes Fichte, Markus Hecher, Stefan Woltran, Markus Zisser: Weighted Model Counting on the GPU by Exploiting Small Treewidth. Submitted Draft.

References

- Günther Charwat, Stefan Woltran: Expansion-based QBF Solving on Tree Decompositions. RCRA@AI*IA 2017: 16-26
- Johannes Fichte, Markus Hecher, Michael Morak and Stefan Woltran: Exploiting Treewidth for Projected Model Counting and its Limits. SAT 2018. To appear.
- Johannes Fichte, Markus Hecher, Stefan Woltran, Markus Zisser: Weighted Model Counting on the GPU by Exploiting Small Treewidth. Submitted Draft.

Thanks for your attention ;)