Weighted Model Counting on the GPU by Exploiting Small Treewidth

Johannes K. Fichte[1] Markus Hecher[2,3] Stefan Woltran[2] Markus Zisser[2]

[1] TU Dresden, Germany
[2] TU Wien, Austria
[3] University of Potsdam, Germany

Shonan Meeting on Parameterized Graph Algorithms & Data Reduction,
Shonan, Japan

March 7th, 2019

Motivation

Model Counting (#SAT)

- Generalizes Boolean satisfiability problem (SAT)
- #SAT: output the number of satisfying assignments
- Various applications in AI and reasoning, e.g.,
  - Bayesian reasoning [Sang et al.'05]
  - Learning preference distributions [Choi et al.'15]
  - Infrastructure reliability [Meel et al.17]
- Computational complexity: #P-hard [Roth'96]

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.
Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Input normal form

- Conjunctive normal form (CNF)
- Form: $F = (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \ldots \wedge (\ldots)$ where ℓ_i either x or $\neg x$

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.
Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Input normal form

- Conjunctive normal form (CNF)
- Form: $F = (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \ldots \wedge (\ldots)$ where ℓ_i either x or $\neg x$

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.

Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

▸ Skip Example

Example

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

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

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given:	Propositional formula F.
Question:	Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Example

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

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

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.

Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Example

$$\overset{1}{(a \vee \neg b \vee d)} \wedge (\neg a \vee \neg c \vee \neg d) \wedge$$

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

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.

Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Example

$$\underbrace{(a \vee \neg b \vee d)}_{1} \wedge \underbrace{(\neg a \vee \neg c \vee \neg d)}_{1} \wedge$$

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

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

SAT-Problem (Boolean Satisfiability Problem)

Given:	Propositional formula F.
Question:	Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

▸ Skip Example

Example

$$\cancelto{1}{(a \vee \neg b \vee d)} \wedge \cancelto{1}{(\neg a \vee \neg c \vee \neg d)} \wedge$$

$$\cancelto{1}{(\neg a \vee \neg d)} \wedge (b \vee c) \wedge (d \vee e) \wedge (\neg b \vee \neg e)$$

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given:	Propositional formula F.
Question:	Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Example

$$\cancelto{1}{(a \vee \neg b \vee d)} \wedge \cancelto{1}{(\neg a \vee \neg c \vee \neg d)} \wedge$$
$$\cancelto{1}{(\neg a \vee \neg d)} \wedge \cancelto{1}{(b \vee c)} \wedge (d \vee e) \wedge (\neg b \vee \neg e)$$

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.

Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Example

$$(a \vee \neg b \vee d)^{1} \wedge (\neg a \vee \neg c \vee \neg d)^{1} \wedge$$
$$(\neg a \vee \neg d)^{1} \wedge (b \vee c)^{1} \wedge (d \vee e)^{1} \wedge (\neg b \vee \neg e)$$

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.

Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

Skip Example

Example

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

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.

Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

▸ Skip Example

Example

$$\overset{1}{(a \lor \neg b \lor d)} \land \overset{1}{(\neg a \lor \neg c \lor \neg d)} \land$$
$$\overset{1}{(\neg a \lor \neg d)} \land \overset{1}{(b \lor c)} \land \overset{1}{(d \lor e)} \land \overset{1}{(\neg b \lor \neg e)}$$

$\Rightarrow$ Satisfiable

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

Given: Propositional formula F.

Question: Is there a truth assignment τ to the variables in F such that F_τ evaluates to 1 (satisfiable).

▸ Skip Example

Example

$$(a \lor \neg b \lor d) \land (\neg a \lor \neg c \lor \neg d) \land$$
$$(\neg a \lor \neg d) \land (b \lor c) \land (d \lor e) \land (b \lor \neg e)$$

#SAT (Number SAT)

- Number of of satisfying truth assignments to F.

Problem of Interest cont.

Weighted Model Counting (WMC)

- Generalizes #SAT
- Given Boolean formula, e.g., $F = (x \vee y) \wedge (\neg x \vee \neg y)$

  Function that maps literals to reals between 0 and 1, e.g.,

  $x \mapsto 0.4$, $\neg x \mapsto 0.6$, $y \mapsto 0.7$, $\neg y \mapsto 0.3$
- Weight of an assignment α is the product over the weights of its literals, i.e., $w(\alpha) := \Pi_{v \in \alpha^{-1}(1)} w(v) \cdot \Pi_{v \in \alpha^{-1}(0)} w(\neg v)$,

  e.g., $\alpha(x) = 1$, $\alpha(y) = 0 \Rightarrow w(\alpha) = 0.4 \cdot 0.3 = 0.16$
- Weighted model count (WMC) of formula is the sum of weights over all its satisfying assignments, e.g.,

  $w(F) = w(\alpha_1) + w(\alpha_2) = 0.12 + 0.42 = 0.54$

Problem of Interest cont.

Weighted Model Counting (WMC)

- Generalizes #SAT
- Given Boolean formula, e.g., $F = (x \vee y) \wedge (\neg x \vee \neg y)$

  Function that maps literals to reals between 0 and 1, e.g.,

  $x \mapsto 0.4$, $\neg x \mapsto 0.6$, $y \mapsto 0.7$, $\neg y \mapsto 0.3$
- Weight of an assignment α is the product over the weights of its literals, i.e., $w(\alpha) := \Pi_{v \in \alpha^{-1}(1)} w(v) \cdot \Pi_{v \in \alpha^{-1}(0)} w(\neg v)$,

  e.g., $\alpha(x) = 1$, $\alpha(y) = 0 \Rightarrow w(\alpha) = 0.4 \cdot 0.3 = 0.16$
- Weighted model count (WMC) of formula is the sum of weights over all its satisfying assignments, e.g.,

  $w(F) = w(\alpha_1) + w(\alpha_2) = 0.12 + 0.42 = 0.54$

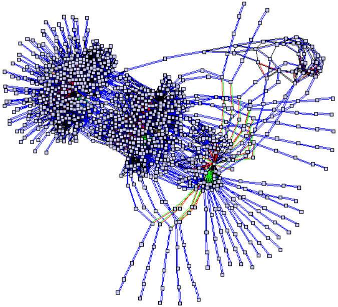
Motivation: What's the issue?

Theory:

SAT cannot be solved faster than $2^{o(n)}$ steps! (ETH)

Idea:

- Practical instances are usually highly structured
- Structure can be exploited by algorithms



$\Rightarrow$ Various Approaches in Solvers to exploit Structure

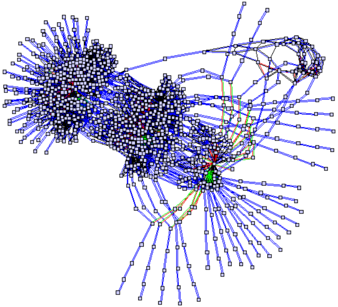
Motivation: What's the issue?

Theory:

SAT cannot be solved faster than $2^{o(n)}$ steps! (ETH)

Idea:

- Practical instances are usually highly structured
- Structure can be exploited by algorithms



$\Rightarrow$ Various Approaches in Solvers to exploit Structure

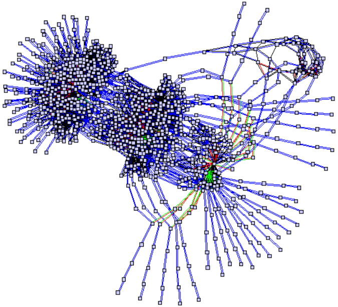
Motivation: What's the issue?

Theory:

SAT cannot be solved faster than $2^{o(n)}$ steps! (ETH)

Idea:

- Practical instances are usually highly structured
- Structure can be exploited by algorithms



$\Rightarrow$ Various Approaches in Solvers to exploit Structure

Motivation: A somewhat different approach.

#SAT/WMC Solving

- There are already various solvers based on various techniques: approximate (Meel) / CDCL (Baccus/Thurley) / knowledge compilation based (Darwiche et al.)

Parameterized Algorithms

- Lots a theoretical work over last 20 years and various algorithms for #SAT

Research Question

Are (theoretical) algorithms from parameterized complexity even useful for implementations in #SAT/WMC solving?

Motivation: A somewhat different approach.

#SAT/WMC Solving

- There are already various solvers based on various techniques: approximate (Meel) / CDCL (Baccus/Thurley) / knowledge compilation based (Darwiche et al.)

Parameterized Algorithms

- Lots a theoretical work over last 20 years and various algorithms for #SAT

Research Question

Are (theoretical) algorithms from parameterized complexity even useful for implementations in #SAT/WMC solving?

Topic of the Talk

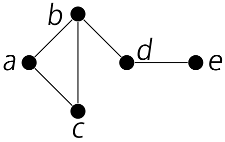
Solve #SAT/WMC by means of an implementation of a parameterized algorithm that exploits small treewidth.

Tree Decompositions

Treewidth ‣ Definition & Example

- Most prominent graph invariant
- Small treewidth indicates tree-likeness and sparsity
- Can be used to solve #SAT/WMC by defining graph representations of the input formula

Tree Decompositions

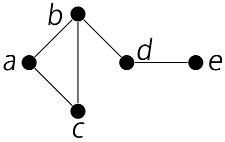


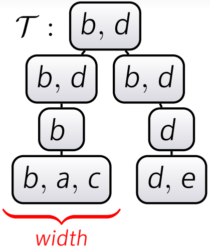
Treewidth

Definition & Example

- Treewidth defined in terms of tree decompositions (TD)
- TD: arrangement of graph into a tree + bags s.t. ...
- Treewidth: width of a TD of smallest width

Tree Decompositions



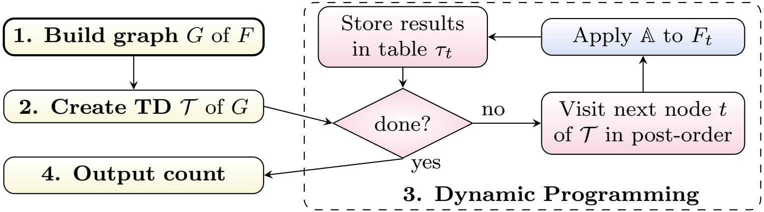


Treewidth

Definition & Example

- Treewidth defined in terms of tree decompositions (TD)
- TD: arrangement of graph into a tree + bags s.t. ...
- Treewidth: width of a TD of smallest width

Outline



Part:

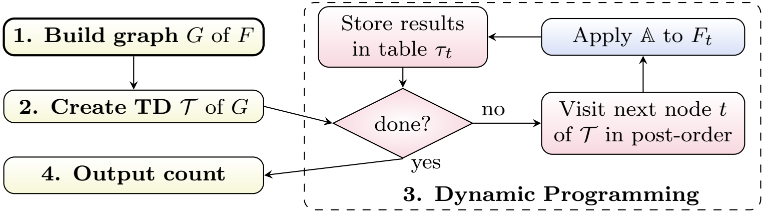
A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

Outline



Part:

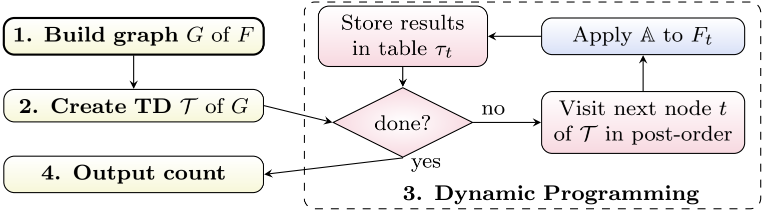
A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

Outline



Part:

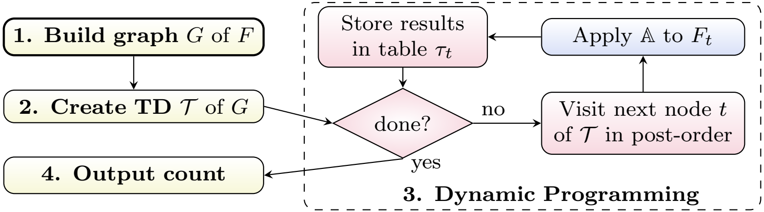
A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

Outline



Part:

A) Background & Basic Concepts

Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA'10]

B) Finding TDs (2)

C) Dynamic Programming (3) on the GPU

“Find” tree decompositions of small width?

Works well even for relatively large instances.

Thanks to the Parameterized Algorithms and Computational Experiments Challenge (PACE) ’16/’17!!!

“Find” tree decompositions of small width?

Works well even for relatively large instances.

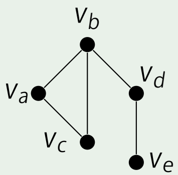
Thanks to the Parameterized Algorithms and Computational Experiments Challenge (PACE) ’16/’17!!!

How to “use” tree decompositions for #SAT/WMC?

By Example

$v_a \vee v_b.\ v_a \vee v_c.$
$v_b \vee v_c.\ v_b \vee v_d.$
$v_d \vee v_e.$

Formula F



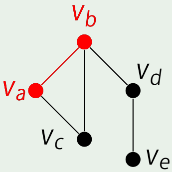
Primal graph

Graph Representations

By Example

$v_a \vee v_b.\ v_a \vee v_c.$
$v_b \vee v_c.\ v_b \vee v_d.$
$v_d \vee v_e.$

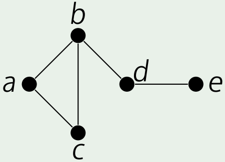
Formula F



Primal graph

Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]

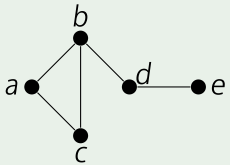


1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

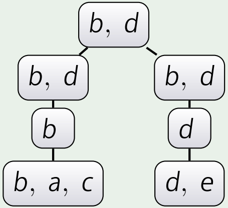
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



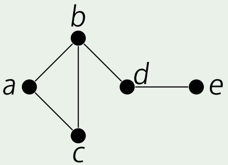
1. Decompose graph
2. Algorithm for SAT
3. Combine solutions



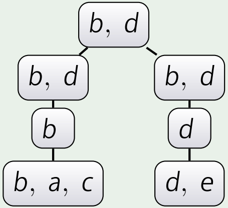
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



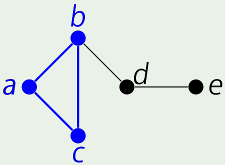
1. Decompose graph
2. Algorithm for SAT
3. Combine solutions



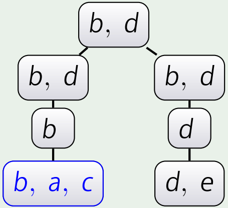
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

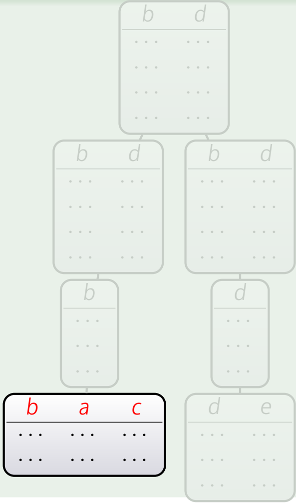
Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

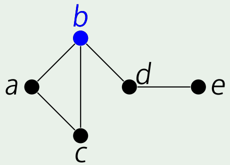




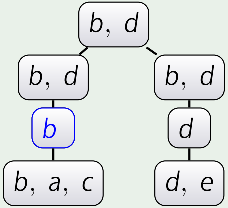
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

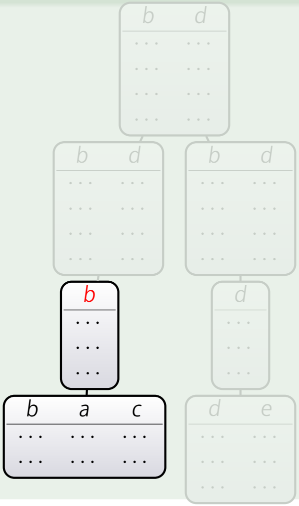
Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

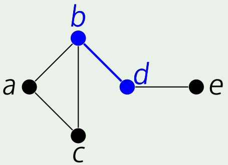




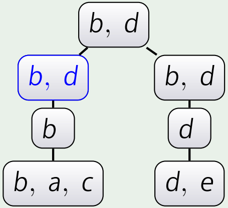
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

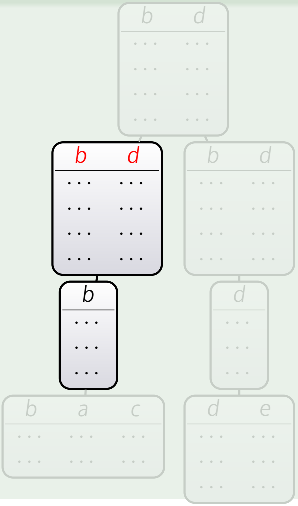
Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

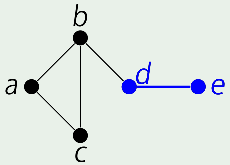




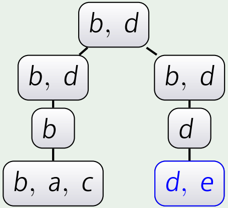
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

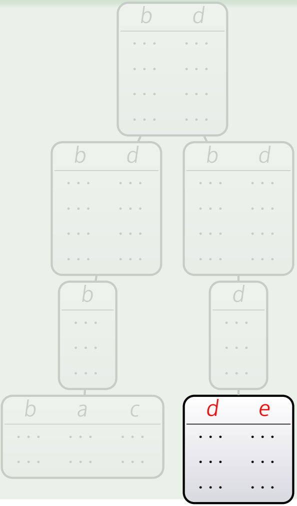
Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



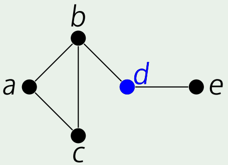
1. Decompose graph
2. Algorithm for SAT
3. Combine solutions



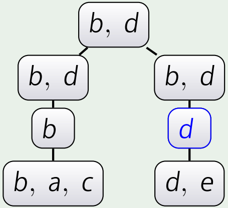


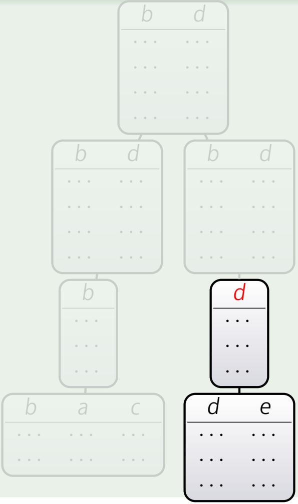
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

Dynamic Programming for SAT [Samer & Szeider'10]



1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

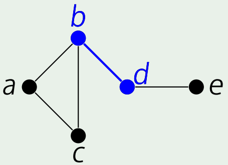




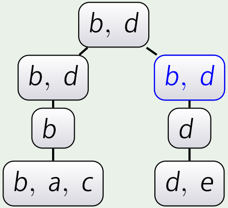
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

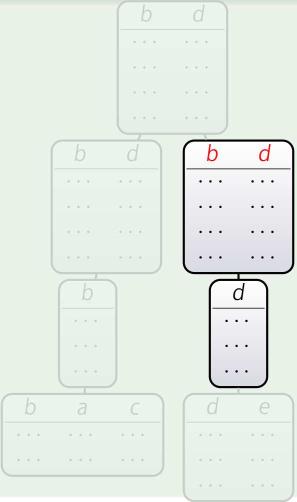
Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



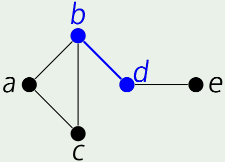
1. Decompose graph
2. Algorithm for SAT
3. Combine solutions



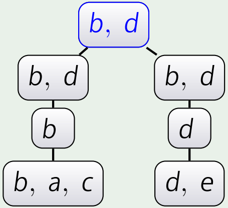


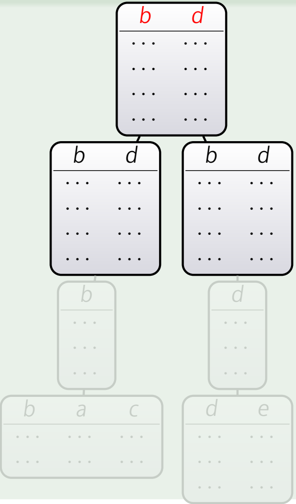
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

Dynamic Programming for SAT [Samer & Szeider'10]



1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

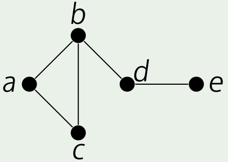




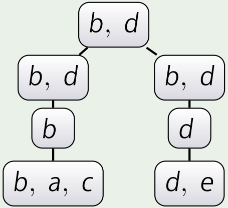
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

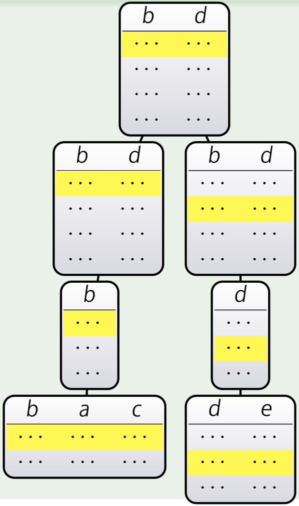
Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

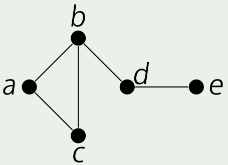




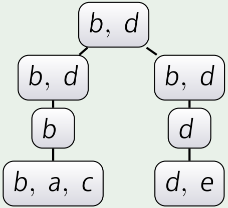
Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

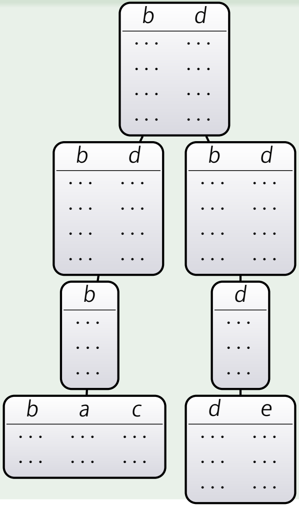
Exploiting Tree Decompositions (TDs)

Dynamic Programming for SAT [Samer & Szeider'10]



1. Decompose graph
2. Algorithm for SAT
3. Combine solutions

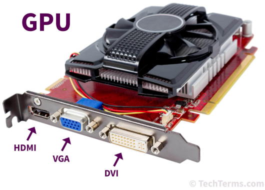




Runtime: $\mathcal{O}(2^k \cdot \|F\|^2)$ where F is the input formula and k the width of TD

A GPU-based #SAT/WMC-solver

OR how to go parallel?



Dynamic Programming on the GPU

How to parallelize DP?

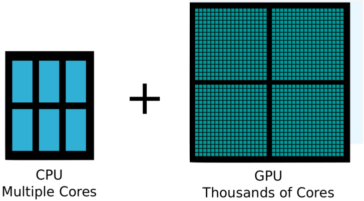
1. Compute tables for multiple nodes in parallel

⇒ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

⇒ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

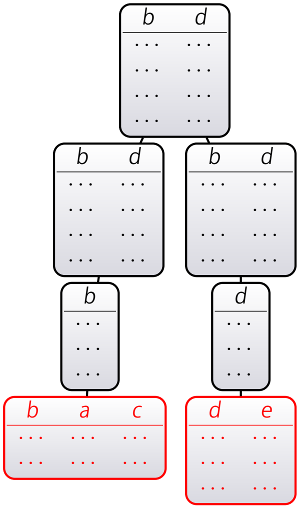
1. Compute tables for multiple nodes in parallel

⇒ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

⇒ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

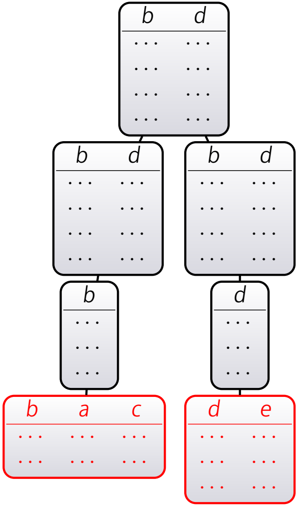
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

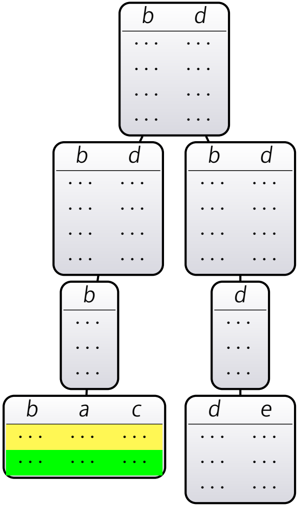
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Dynamic Programming on the GPU

How to parallelize DP?

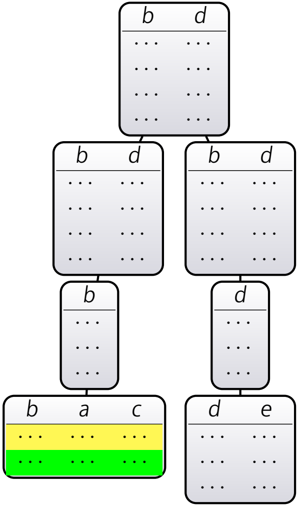
1. Compute tables for multiple nodes in parallel

$\Rightarrow$ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units

$\Rightarrow$ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



Implementation (gpuSAT1)

Right hindsight?

- Data structures: a "pixel" represents #solutions (store data as array)
- Table merging: merge small bags (< 14)
- Table splitting: split large tables
- Weight: avoid double overflows by factor for #SAT
  (so it's actually WMC with uniform weights).

Implementation

- OpenCL: vendor and hardware independent computation framework; C++11
- Works for two graph types: primal and incidence graph
- Supports weighted model counting (WMC)

Experimental Work (Naive Implementation)

Instances

- 2585 instances from public benchmarks
- #SAT and WMC

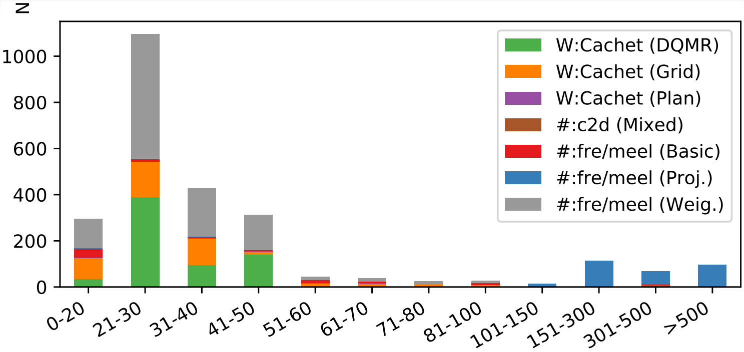
Limits

- Cannot expect to solve instances of high treewidth.

Hardware

- non-GPU solving: cluster of 9 nodes; each 2xE5-2650 CPUs(12cores) 2.2 GHz, 256 GB RAM; disabled HT, kernel 4.4
- GPU-solving: i3-3245 3.4 GHz; 16 GB RAM; GPU: Sapphire Pulse ITX Radeon RX 570 GPU; 1.24 GHz with 32 compute units, 2048 shader units, 4GB VRAM

Distribution of Primal Width



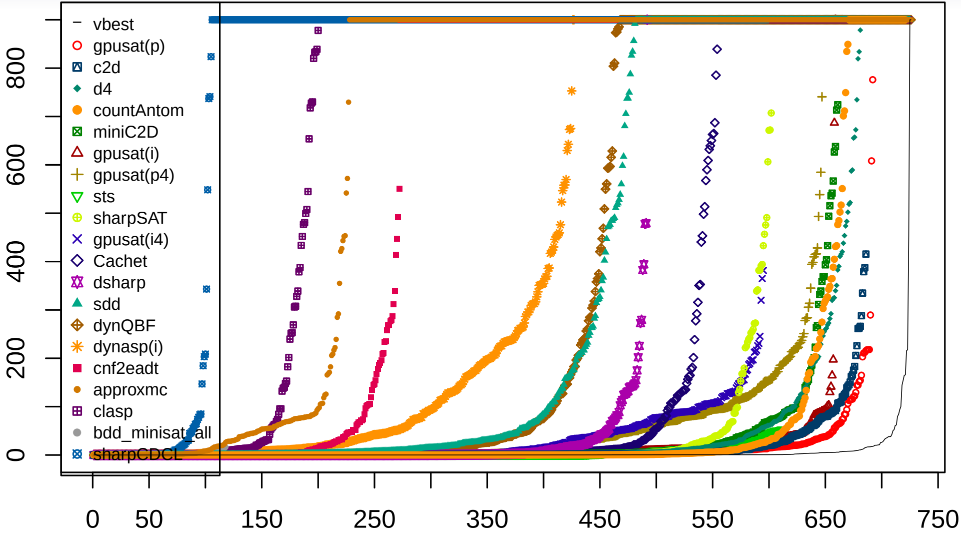
Decomposition Heuristic:

- Runtime well below a second (max. 2.5) 0–40
- Timeout (900s) on 41 instances

$\Rightarrow$ 54% primal treewidth below 30; 70% below 40

Parameterized Algorithms might work...

Solving (Width: 0–30): #SAT



Solving: #SAT

solver	0-20	21-30	31-40	41-50	51-60	>60	best	$\sum$	rank
c2d	164	519	**175**	116	20	118	120	1112	2
Cachet	133	421	91	109	8	58	13	820	7
d4	**169**	510	156	119	**23**	**162**	191	**1139**	1
gpusat(p)	**169**	**523**	79	104	0	0	88	875	6
miniC2D	167	491	137	103	8	67	2	973	4
sharpSAT	136	465	136	112	11	124	**483**	984	3
sts	162	448	101	**146**	10	45	252	912	5

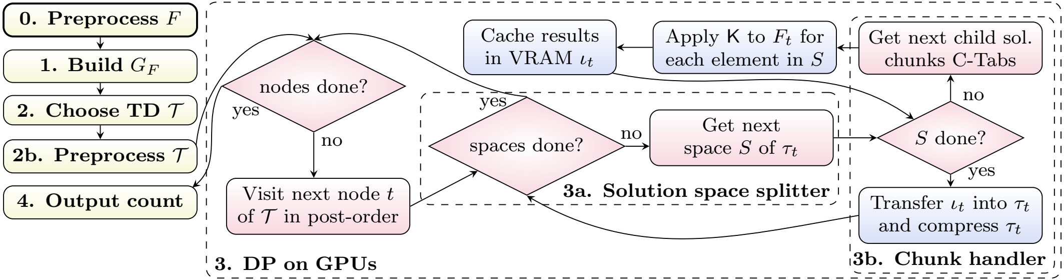
Table: Number of counting instances solved by solver and interval.

Empirical Work (first approach)

Observations

- Implementation is fairly naive
- Still: competitive up to width 30
- Requirement: obtain decompositions fast
- Width was surprisingly small (different for SAT)

New Architecture

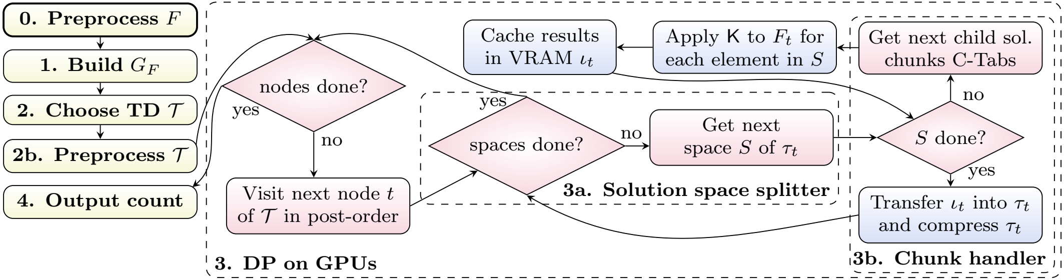


0. Instance Preprocessing
2. Customized Tree Decompositions

3a. Solution Space Splitting

3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
Compress the data and store it in the VRAM (separate GPU-programs)
After all chunks are processed memory regions are merged

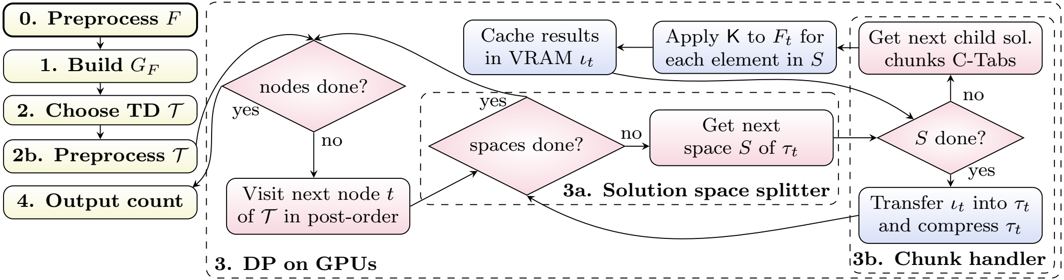
New Architecture



0. Instance Preprocessing
2. Customized Tree Decompositions
   (#30; minimize max. card. of intersection of bags at node and its children)

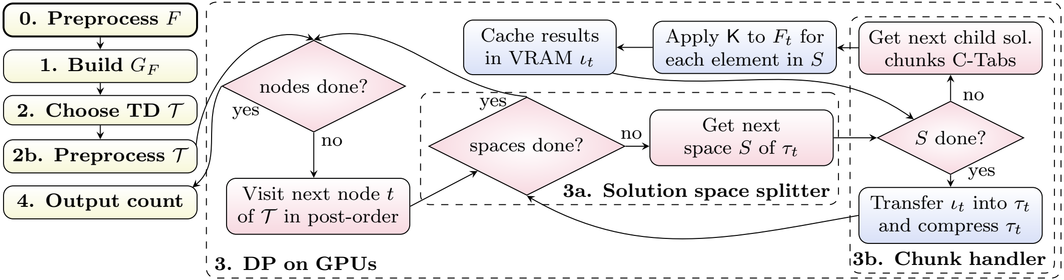
3a. Solution Space Splitting
3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
   Compress the data and store it in the VRAM (separate GPU-programs)
   After all chunks are processed memory regions are merged

New Architecture



0. Instance Preprocessing
2. Customized Tree Decompositions
3a. Solution Space Splitting
(Split larger solutions into smaller portions $\Rightarrow$ avoid OOM)
3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
Compress the data and store it in the VRAM (separate GPU-programs)
After all chunks are processed memory regions are merged

New Architecture



0. Instance Preprocessing
2. Customized Tree Decompositions

3a. Solution Space Splitting

3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
    Compress the data and store it in the VRAM (separate GPU-programs)
    After all chunks are processed memory regions are merged

Data Structures (Save Space)

Disclaimer for theorists: you need to get your hands dirty
(essentially: bit fiddling)

Data Structures (Save Space)

1. Store compressed partial assignments
   (only where # $\neq 0$, simulate a BST in an array)
2. Use logarithmic counters

Data Structures (Save Space)

1. Store compressed partial assignments
   (only where # $\neq 0$, simulate a BST in an array)
2. Use logarithmic counters

1) Assignment Compression (BST in an Array)

- Continuous sequence 64-bit unsigned integers (cells)
- Cell: empty, index, and value (counter)
- index cells: lower 32 bits index to the next cell (variable $\rightarrow$ 0), upper bits (1)
- Handle Sync by keeping track of the current size
  (number of allocated cells; prevent to allocate cell again)

Data Structures (Save Space)

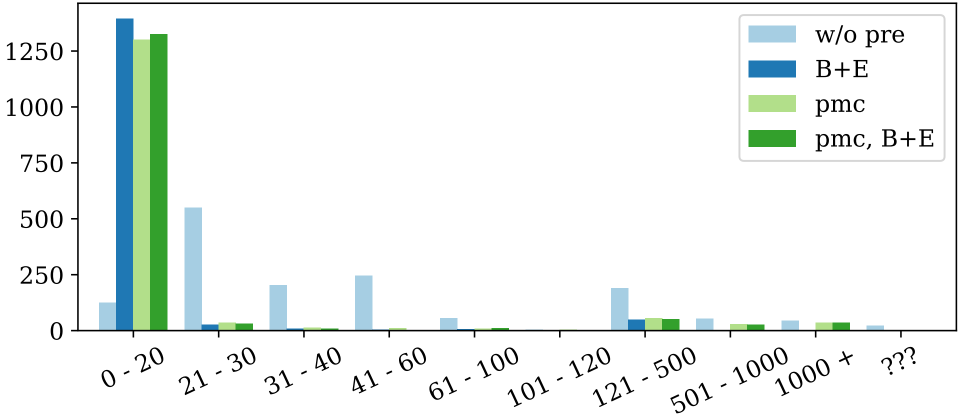
1. Store compressed partial assignments
   (only where # $\neq 0$, simulate a BST in an array)
2. Use logarithmic counters

2) Data Type Precision

- Store floating log-counters
- Numbers stored in relation to exponent 2^{e} (largest exponent)
- Dynamically change exponent (keep highest possible precision)

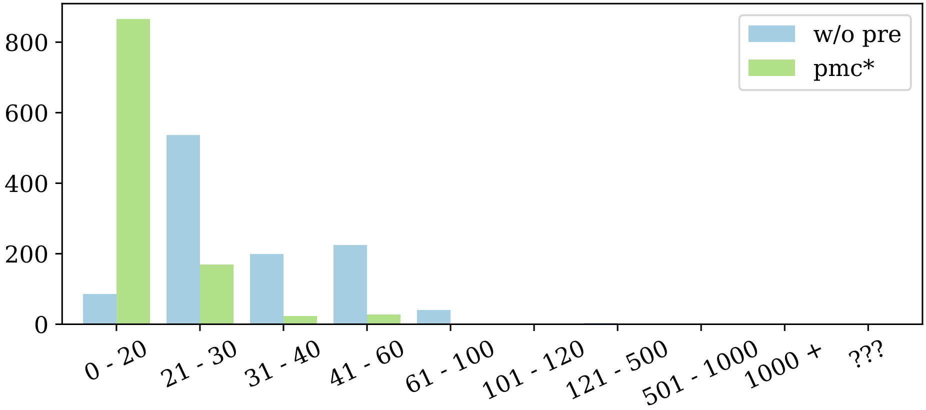
Where are we at with the new architecture?

#SAT: Width Comparison (w/o Preprocessing)



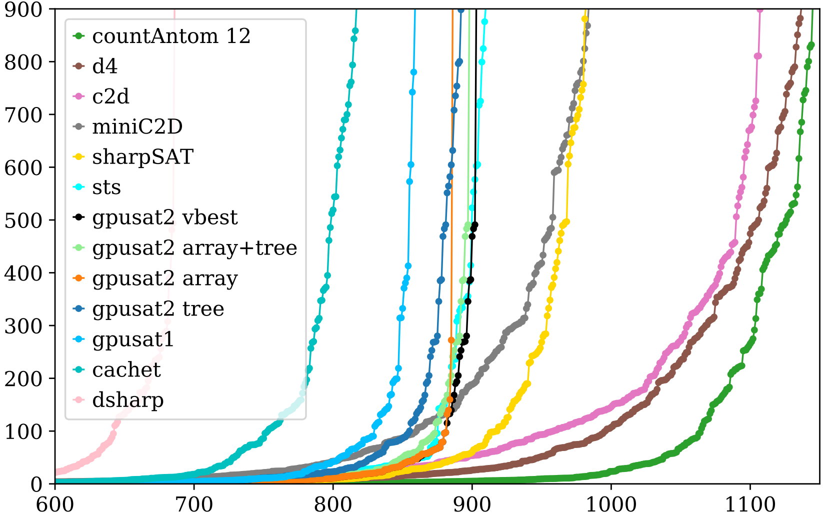
⇒ Produce decompositions of significantly smaller width

WMC: Width Comparison (w/o Preprocessing)

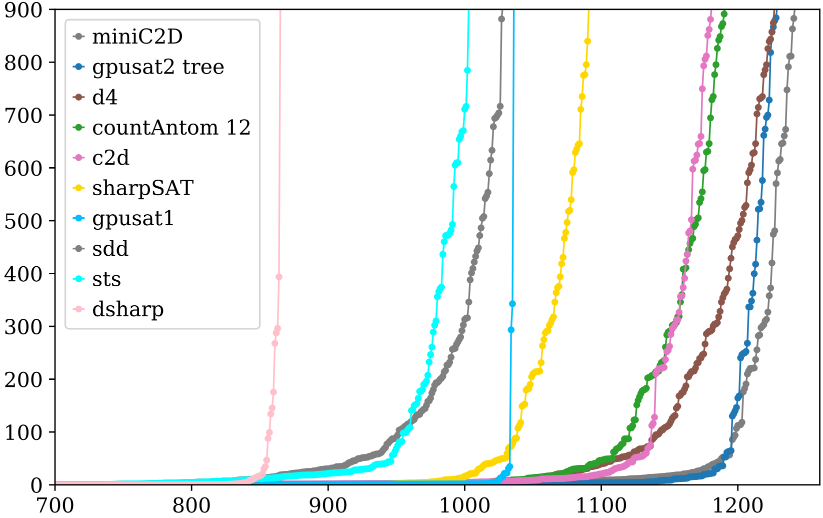


$\Rightarrow$ Produce decompositions of significantly smaller width

#SAT: Runtime Results (wo. Preprocessing)

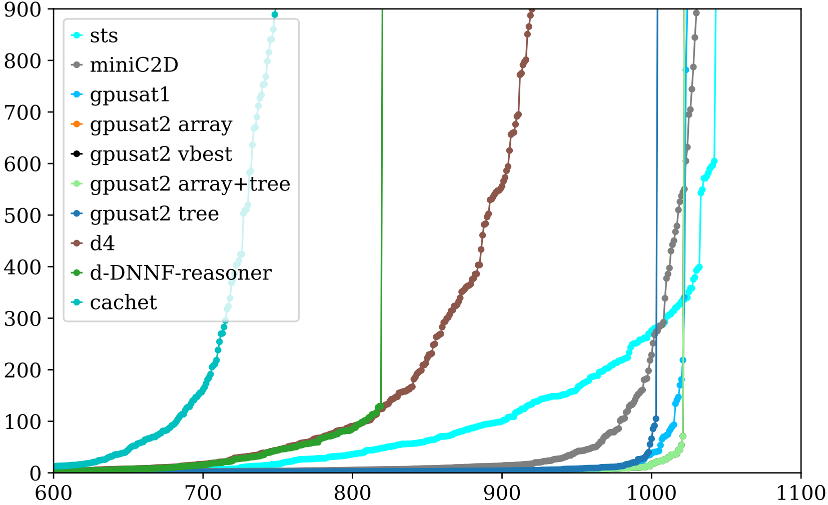


#SAT: Runtime Results (w. Preprocessing)



⇒ Techniques pay off after preprocessing

#WMC: Runtime Results (w. Preprocessing)



⇒: After preprocessing #SAT no3, WMC no2

Summary

Contributions

- Established Architecture for DP on the GPU
- Competitive Implementation for #SAT/WMC solving

Benchmark: Comparing apples and oranges

BUT: you compare parallel and sequential solvers.

1. We run on cheap consumer hardware (200 EUR).
2. Cannot measure speedup due to OpenCL limitations
   $\Rightarrow$ migrate to cuda

Summary contd.

Take Home Messages

1. Parameterized Algorithms can actually work
   (Preprocessing is key; some techniques pay only off with right preprocessing)
2. Does it work for SAT? $\Rightarrow$ we don't expect so.

Future Work

- Improve current setup by:
  Portfolio solving; Parallel Usage of GPUs; Alternative Frameworks
- Parameters (pswidth)

Sponsors: FWF Y698 & P26696; DFG HO 1294/11-1

Summary contd.

Take Home Messages

1. Parameterized Algorithms can actually work
   (Preprocessing is key; some techniques pay only off with right preprocessing)
2. Does it work for SAT? $\Rightarrow$ we don't expect so.

Future Work

- Improve current setup by:
  Portfolio solving; Parallel Usage of GPUs; Alternative Frameworks
- Parameters (pswidth)

Thanks for listening!

GitHub:daajoe/{benchmark-tool,frasmt,trellis}

Sponsors: FWF Y698 & P26696; DFG HO 1294/11-1