Model Counting: Solving, Complexity, and Applications

Johannes Fichte

PCCR, Lisbon, July 24, 2026

Outline

Preliminaries / Motivation

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem) (Fichte, Berre, Hecher, and Szeider, 2023a)

Given: Propositional formula F.
Question: Can we assign α variables in F to $0/1$ such that F_α evaluates to 1, i.e., satisfiable?

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem) (Fichte, Berre, Hecher, and Szeider, 2023a)

Given: Propositional formula F.
Question: Can we assign α variables in F to $0/1$ such that F_α evaluates to 1, i.e., satisfiable?

Example

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

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem) (Fichte, Berre, Hecher, and Szeider, 2023a)

Given: Propositional formula F.
Question: Can we assign α variables in F to $0/1$ such that F_α evaluates to 1, i.e., satisfiable?

Example (Satisfying Assignment)

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

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem) (Fichte, Berre, Hecher, and Szeider, 2023a)

Given: Propositional formula F.
Question: Can we assign α variables in F to $0/1$ such that F_α evaluates to 1, i.e., satisfiable?

Example (Test Assignment)

$$F = (\neg a \vee b \vee x)^{1} \wedge (a \vee b)^{1} \wedge (c \vee \neg x)^{1} \wedge (b \vee \neg c)^{1} \wedge (\neg b \vee \neg c \vee \neg y)^{1}$$

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem) (Fichte, Berre, Hecher, and Szeider, 2023a)

Given: Propositional formula F.
Question: Can we assign α variables in F to $0/1$ such that F_α evaluates to 1, i.e., satisfiable?

Example

$$F = (\neg a \vee b \vee x) \overset{1}{\wedge} (a \vee b) \overset{1}{\wedge} (c \vee \neg x) \overset{1}{\wedge} (b \vee \neg c) \overset{1}{\wedge} (\neg b \vee \neg c \vee \neg y) \overset{1}{}$$

Theory

- SAT is computationally hard (NP-complete) (Cook, 1971; Levin, 1973)

Problem of Interest

SAT-Problem (Boolean Satisfiability Problem) (Fichte, Berre, Hecher, and Szeider, 2023a)

Given: Propositional formula F.
Question: Can we assign α variables in F to $0/1$ such that F_α evaluates to 1, i.e., satisfiable?

Example (All Models)

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

Practice

- We don't care, we need it.
- Solve instances with millions of variables today (Biere et al., 2021)

Problem of Interest (cont.)

#SAT-Problem (Model Counting)

Given: Propositional formula F.
Question: Output the number of models (total satisfying assignments).
$\mathrm{mc}(F) := |\mathrm{Mod}(F)|$

Example (Same Formula)

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

$$\mathrm{mc}(F) = |\{\{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \{b, c, x\}, \{a, b, c, x\}, \{b, y\}, \{a, b, y\}\}| = 8$$

Problem of Interest (cont.)

#SAT-Problem (Model Counting)

Given: Propositional formula F.
Question: Output the number of models (total satisfying assignments).
$\mathsf{mc}(F) := |\mathsf{Mod}(F)|$

Theory

- MC: Canonical #P-complete problem (Valiant, 1979)
  (Recall: $\mathsf{PH} \subseteq P^{PP} = P^{\#\cdot P}$ where $\bigcup_{k\in\mathbb{N}} \Delta_k^P = \mathsf{PH}$ and $\mathsf{NP} \subseteq \Delta_2^P = P^{\mathsf{NP}}$ cf. (Toda, 1991)
- PMC: $\# \cdot \mathsf{NP}$-c (Durand et al., 2005)
- Fragments: quickly hard, but not systematically studied (under varying reductions)
  More fine-grained complexity (under different calls) (Bannach, Gomez, Demaine, Hecher'25)

Problem of Interest (cont.)

#w-SAT-Problem (Weighted Model Counting)

Given: Propositional formula F and weight function $w : \mathsf{lits}(F) \mapsto \mathbb{Q}$.[a]

Question: Output the weighted model count:
$\mathsf{wmc}(F, w) := \sum_{M \in \mathsf{Mod}(F)} w(M)$ where $w(M) := \prod_{\ell \in \mathsf{lits}(M)} w(\ell)$.

[a] $\mathsf{lits}(F) := \mathsf{var}(F) \cup \{ \neg x : x \in \mathsf{var}(F) \}$ and $\mathsf{lits}(M) := M \cup \{ \neg x : x \in \mathsf{vars}(F) \setminus M \}$

Problem of Interest (cont.)

#w-SAT-Problem (Weighted Model Counting)

Given: Propositional formula F and weight function $w : \mathsf{lits}(F) \mapsto \mathbb{Q}$.
Question: Output the weighted model count.

Algebraic Model Counting

- Natural extension: $\mathbb{C}$
- Define semi-ring even possible and large parts technical to integrate
- Numerical precision might become challenging
- Generally: non-uniform counting results in less preprocessing possibilities

Motivation: Interest in Solving Model Counting

Applications

- Reliability: Software, Power Grids, Satellite Comm (Teuber and Weigl, 2021; Dueñas-Osorio et al., 2017; Bannach et al., 2024)
- Quantum Computing (circuit analysis) (Mei et al., 2024)
- Solution Space Navigation (Fichte et al., 2022a)
- Domain Model Acquisition (Balyo et al., 2024)
- Quantitative vulnerability assessment (Girol et al., 2024)

Competition

Motivation

Why having a solving competition?

Solvers and theoretical concepts improved.

Deepen relationship between latest theoretical and practical development on the various model counting problems and their practical applications.

Goals of the Competition

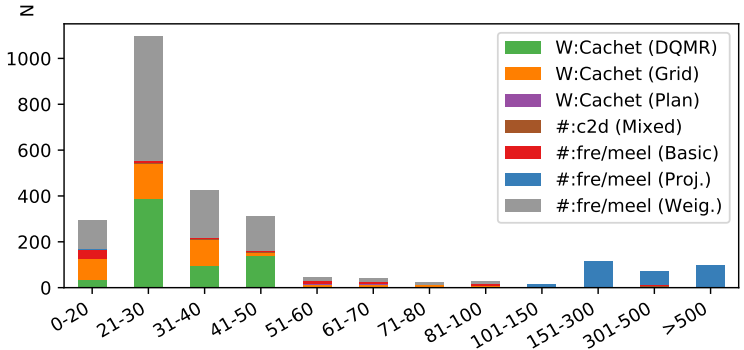
Aims

- Improve solvers
- Identify benchmarks and extend feasibility
- Robustness of solvers
- Exchange of ideas
- Make a large audience of researchers aware of counting techniques

How

- 1st iteration in 2020
- Annual event and results presented at SAT Conference
- Snapshot of the current state-of-the-art

Empirical Work: Distribution of Upper Bounds on Primal Tw (2019)



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

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

Model Counting Competition (cont.)

POS	Submission	#(private)
1	**nus-bareganak**	**75**
2	**nus-narasimha**	**73**
3	**c2d**	**71**
4	nus-onlyapprox	56
5	d4	48
6	swats	33
7	MCSim	23
8	addmc	19
9	ispence	16

2020

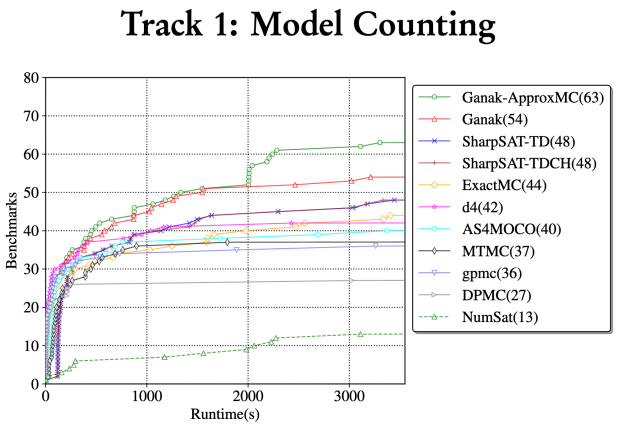
#	Submission	Authors	From	solved	excl
1	**SharpSAT-TD**	Tuukka Korhonen Matti Järvisalo	Helsinki	**78**	0
2	**nus-narasimha (2021)**	Sharma, Lai, Xu, Roy, Yap, Soos, Meel	Singapore, Kanpur, Changchun	**61**	1
3	**d4 (2021)**	Jean-Marie Lagniez Pierre Marquis	Lens	**51**	0
4	gpmc (v1.0.1)	Kenji Hashimoto Takaaki Isogai	Nagoya	38	0
5	MC2021_swats	Sylwester Swat	Poznan	34	0
6	DPMC (2021)	Vu Phan Jeffrey Dudek Moshe Vardi	Huston	34	0
7	c2d (v3.0.0 MC 2021)	Adnan Darwiche	LA	29	0
8	bob	Daniel Pehoushek	Chincinati	11	0
9	SUMC2	Ivor Spence	Belfast	7	0

2021

Model Counting Competition (cont.)

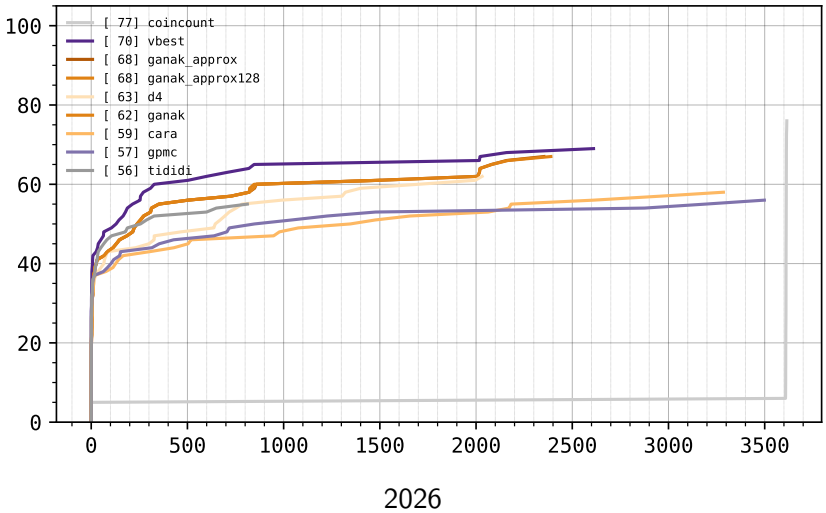
#	Submission	Authors	From	solved	excl
1	**SharpSAT-TD**	Tuukka Korhonen Matti Järvisalo	Helsinki	**78**	0
2	**nus-narasimha (2021)**	Sharma, Lai, Xu, Roy, Yap, Soos, Meel	Singapore, Kanpur, Changchun	**61**	1
3	**d4 (2021)**	Jean-Marie Lagniez Pierre Marquis	Lens	**51**	0
4	gpmc (v1.0.1)	Kenji Hashimoto Takaaki Isogai	Nagoya	38	0
5	MC2021_swats	Sylwester Swat	Poznan	34	0
6	DPMC (2021)	Vu Phan Jeffrey Dudek Moshe Vardi	Huston	34	0
7	c2d (v3.0.0 MC 2021)	Adnan Darwiche	LA	29	0
8	bob	Daniel Pehoushek	Chincinati	11	0
9	SUMC2	Ivor Spence	Belfast	7	0

2021



2024

Model Counting Competition (cont.)



Solving

Modern Solving Techniques

Solvers Model Counting Competitions (Fichte et al., 2021a; Fichte and Hecher, 2025)

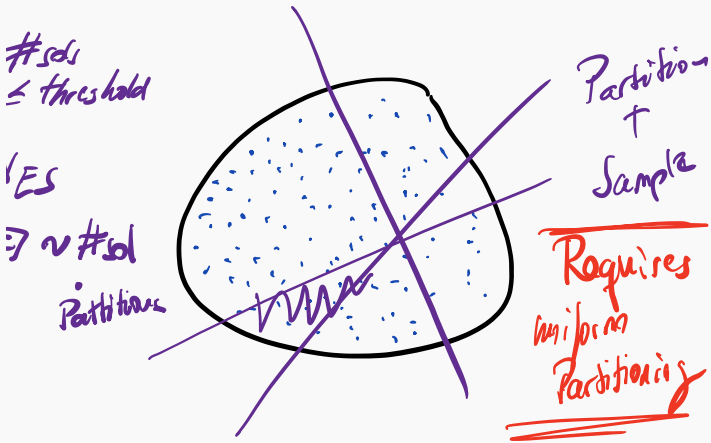
- approximate (Chakraborty et al., 2014)
  (SAT-solver + Gaussian Elimination)
- dynamic programming (Fichte et al., 2019, 2018, 2021b; Dudek et al., 2020)
  (allows massive parallelization or splitting+one of the methods above)
- search (Bacchus et al., 2003; Thurley, 2006; Korhonen and Järvisalo, 2021)
  (modified SAT-solver, component caching, simplification, CDCL)
- knowledge compilation (Oztok and Darwiche, 2015; Lagniez and Marquis, 2017)
  (modified SAT-solver; similar to above, but stores compact representation)

Important Role

- Tree width and tree decompositions

Modern Solving Techniques: Approximate Solving

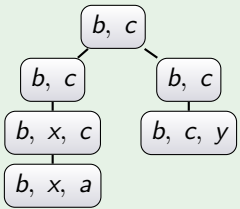
Idea

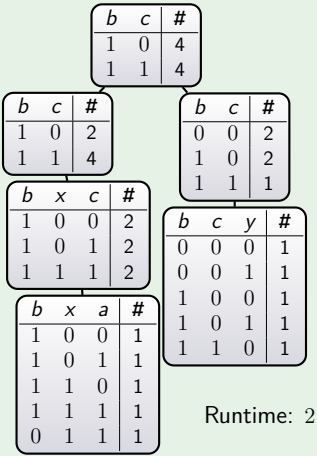


Employing Decompositions for Model Counting

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows





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

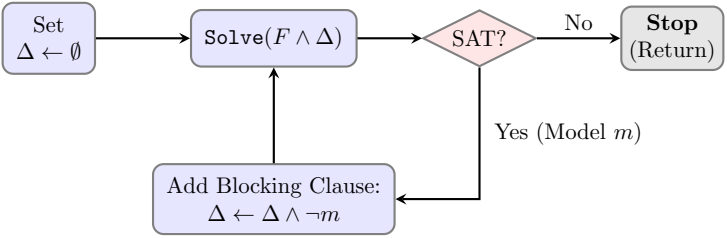
Runtime: $2^{O(\text{treewidth})} \cdot \text{poly}(|F|)$

(#)SAT

width	Solving*	Reference
htw	paraNP-c	Samer, Szeider 2010
undir incidence cw	XP	Ordyniak, Paulusma, Szeider 2010; Slivovsky, Szeider 2013
dir incidence cw	FPT	Courcelle, Makowsky, Rotics 2001
dual tw	FPT $\mathcal{O}^*(2^k)$	Samer, Szeider 2010
incidence tw	FPT $\mathcal{O}^*(4^k)$ $\mathcal{O}^*(2^k)$	Fischer, Makowsky, Ravve 2008; Samer, Szeider 2010 Slivovsky, Szeider 2020
primal tw	FPT $\mathcal{O}^*(2^k)$	Freuder 1985; Alekhnovich, Razborov 2002 Bacchus, Dalamao, Pitassi 2003

Generating: The All-SAT Loop

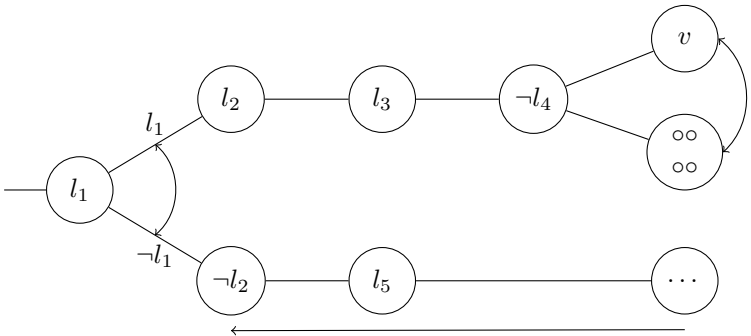
- A naive approach: explicitly listing all satisfying assignments.
- Compute incrementally using a standard SAT solver (Audemard et al., 2013)



- Note: blocking clause $\neg m$ rules out the model m, solver is forced to find new one.

Modern Solving Techniques: Component Caching

- Systematic Search Space Splitting + CDCL
- Split formula systematically by assigning variables
- Component of formula F set C where $\mathsf{vars}(C) \cap \mathsf{vars}(F - C) = \emptyset$
- Store values for computed components



Modern Solving Techniques: Component Caching (cont.)

Component Caching (Sang et al., 2004; Lagniez and Marquis, 2021)

Encounter the same sub-problems (components) in different branches of the search.
Caching: store results to avoid re-solving them

- **The Caching Scheme (r_c):** A mapping that converts a sub-formula G into a **Cache Key** (signature).

$$G \xrightarrow{r_c} \text{Key}$$

- **The Cache:** A hash map storing: $\text{Key} \mapsto \texttt{cnt}$.
- **Correctness Condition:** If two sub-formulas generate the same key, they must be logically equivalent.

$$r_c(G_1) = r_c(G_2) \supset G_1 \equiv G_2$$

Modern Solving Techniques: Component Caching (cont.)

Which Clauses

- **Scheme A: "Standard" All Clauses (cachet)** (Sang et al., 2004)
  - Include **all** clauses belonging to the current component.
  - Safe but verbose.
  - Requirement: Variables are implicit, explicit store of variable set
- **Scheme B: "Non-Binary" (sharpSAT)** (Thurley, 2006)
  - Include only clauses of **size** > 2.
  - Rationale: Binary clauses are numerous but imply little structure. Excluding them saves memory and increases cache hits (abstraction).
  - Requirement: Need to explicitly store which variables are involved.
- **Scheme C: "Not Touched" (D4)** (Lagniez and Marquis, 2017)
  - Include clauses that have been **modified** (shortened) by the current partial ass.
  - Rationale: If a clause is "untouched" (original state), it is implicitly defined by the set of variables involved in the component.
  - Requirement: Need to explicitly store which variables are involved.

Modern Solving Techniques: Component Caching (cont.)

Cache Management: The Cleaning Problem

The cache grows indefinitely.

- Storing millions of components consumes gigabytes of RAM.
- Hash collisions increase, slowing down lookups.

The Strategy

We need a **Cleaning Policy** to periodically remove "low-value" entries.

- Cachet (Age-Based): Old entries are useless
- SharpSAT (Activity-Based): Used entries are useful.
- D4 (Size-Aware): Small components (few variables) are hit more often than large ones. Requires ratio (hits / component size)

Modern Solving Techniques: Heuristics

Classical SAT (Goal)

Pick variable most likely to lead to conflict (learn clauses fast) or to satisfy the formula

- **MOMS** (Maximum Occurrence in cls of Minimum Size) (Crawford and Auton, 1996)
- **VSIDS** (Variable State Independent Decaying Sum) (Moskewicz et al., 2001)

The Counting Dilemma

- **SAT solvers** want conflicts (to prove UNSAT).
- **Counters** want **decomposition** (to multiply sub-counts).
- Pure VSIDS might stick to one giant component for too long.

Modern Solving Techniques: Heuristics (cont.)

Counting Heuristics

- **Cachet/SharpSAT**: (Sang et al., 2004)
  - A hybrid score combining activity and structural relevance
    Variable State Aware Decaying Sum (VSADS)
    $VSADS(x) = p \cdot \text{VSIDS}(x) + q \cdot \text{"MOMS"}(x)$
- **ApproxMC/Ganak**: (Sharma et al., 2019b)
  - **Component Caching** is the primary speedup mechanism in counting
  - Branch on variables that define cache keys, increase probability of a **Cache Hit**
    Cache State and Variable State Aware Decaying Sum (CSVSADS).
- **C2D** (Darwiche, 2004)
  - Force solver to follow a **pre-computed roadmap** that guarantees decomposition.
- **D4**: integrates graph partitioning directly into the search loop (KaHyPar)
- **sharpSAT-TD** (Korhonen and Järvisalo, 2021)
  - VSADS + tw penalty; Tree Decomposition (TD) soft guide

Modern Solving: Knowledge Compilation

Bottleneck

- SAT and #SAT are NP-complete and #P-complete. In practice, there is no guarantee of solving queries within a strict time limit.

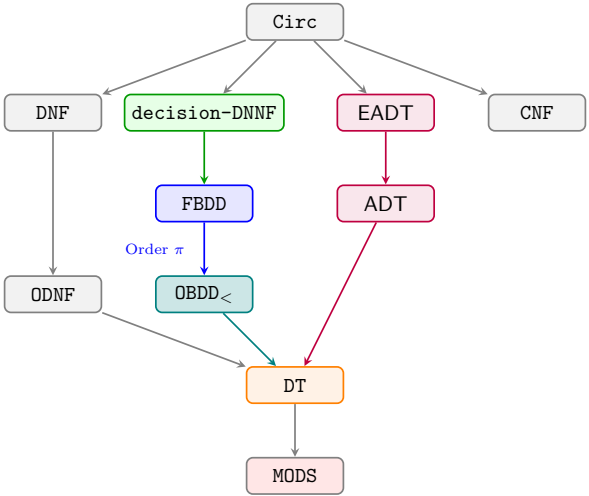
The KC Paradigm: Split the Effort

1. **Offline Phase (Compilation):** Translate the instance Σ into a representation $\mathcal{C}$ from a target language $\mathcal{L}$. This step is computationally expensive.
2. **Online Phase (Querying):** Use $\mathcal{C}$ to answer difficult queries (like Model Counting) in **polynomial time**.

When useful?

- Offline cost is amortized over many queries on the same fixed knowledge base.
- **The Question:** Which language $\mathcal{L}$ should we choose? We use the Knowledge Compilation Map.

Modern Solving: Knowledge Compilation (cont.)



Modern Solving: Knowledge Compilation (cont.)

Evaluating a Language $\mathcal{L}$

To choose a language, we evaluate its tractability on two axes:

1. Queries
(Extracting Information)

- **CO** (Consistency)
- **CE** (Clause Entailment)
- **VA** (Validity)
- **EQ** (Equivalence)
- **SE** (Sentential Entailment)
- **IM** (Implicant Checking)
- **CT** (Model Counting)
- **ME** (Model Enumeration)

2. Transformations
(Modifying the Knowledge)

- **CD** (Conditioning)
- $\wedge$**C**, $\wedge$**BC** (Closure under $\wedge$)
- $\vee$**C**, $\vee$**BC** (Closure under $\vee$)
- $\neg$**C** (Closure under $\neg$)
- **FO**, **SFO** (Forgetting)

Modern Solving: Knowledge Compilation (cont.)

$\mathcal{L}$	CO	VA	CE	IM	EQ	SE	CT	ME
Circ	○	○	○	○	○	○	○	○
CNF	○	√	○	√	○	○	○	○
DNF	√	○	√	○	○	○	○	√
ODNF	√	√	√	√	√	√	√	√
MODS	√	√	√	√	√	√	√	√
decision − DNNF	√	√	√	√	?	○	√	√
FBDD	√	√	√	√	○	○	√	√
DT	√	√	√	√	√	√	√	√
EADT	√	√	√	√	?	○	√	√
ADT	√	√	√	√	√	√	√	√
$\text{OBDD}_{<}$	√	√	√	√	√	√	√	√
SDD (Canon.)	√	√	√	√	√	√	√	√
SDD	√	√	√	√	√	√	√	√

Modern Solving: Knowledge Compilation (cont.)

$\mathcal{L}$	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
Circ	√	○	√	√	√	√	√	√
CNF	√	○	√	√	√	○	√	○
DNF	√	√	√	○	√	√	√	○
ODNF	√	○	○	○	√	○	○	○
MODS	√	√	√	○	√	○	√	○
decision − DNNF	√	○	○	○	○	○	○	?
FBDD	√	○	○	○	○	○	○	√
DT	√	○	√	○	√	○	√	√
EADT	√	○	○	○	○	○	○	√
ADT	√	○	√	○	√	○	√	√
$\text{OBDD}_{<}$	√	○	√	○	√	○	√	√
SDD (Canon.)	○	○	○	○	○	○	○	√
SDD	√	○	√	○	√	○	√	√

Modern Solving: Knowledge Compilation (cont.)

Parameterized View

- on Tree Decompositions (incidence) construct d-DNNFs by (Bova et al., 2015)
- on Path Decompositions construct OBDDs by (Amarilli et al., 2020)
- (indirect results) about treewidth and SDDs by (Bova and Szeider, 2017)

Application (Planning): Decisions and Solution Spaces

AI Planning

Planning Task

- Task $\Pi = \langle \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ where
  - $\mathcal{A}$: finite set of propositional state variables.
  - $\mathcal{O}$ finite set of operators (preconditions and effects)
  - $\mathcal{I}$: initial state of Π
  - $\mathcal{G}$: goal condition
- State s is a mapping $s : \mathcal{A} \to \{0, 1\}$ and Goal state if $s_* \models \mathcal{G}$.

Plan

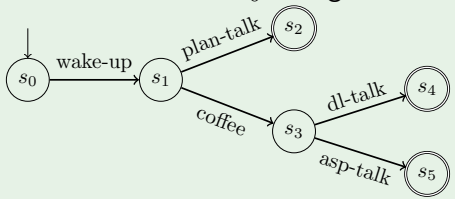
- Sequence $\pi = \langle o_0, \ldots, o_{n-1} \rangle$ is a of applicable operators
- Generates a sequence of states $s_0, \ldots, s_n$, where $s_0 = \mathcal{I}$, s_n is a goal state
- Assumption: Poly-bounded plan length

AI Planning (cont.)

(Speck, Hecher, Gnad, Fichte, and Corrêa, 2025)
(Gnad, Hecher, Gaggl, Rusovac, Speck, and Fichte, 2025)

Example: Chaotic Researcher attending KR

State space. The initial state s_0; the goal states s_2, s_4, s_5.



- Enumerate: List all solutions?

$\mathrm{Sol}(P) = \{\{\text{wake-up}, \text{plan-talk}\}, \{\text{wake-up}, \text{coffee}, \text{dl-talk}\}, \{\text{wake-up}, \text{coffee}, \text{asp-talk}\}\}$

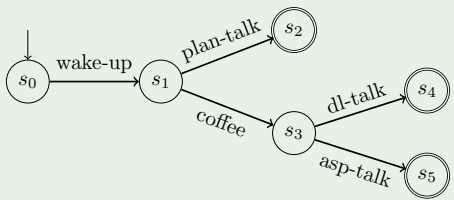
- Counting: How many solutions does the problem have? $|\mathrm{Sol}(P)| = 3$
- Projected Count: How many solutions does the problem have when interested only in coffee? $\{\{\}, \{\text{coffee}\}\}$ 2

Counting to Explain Plan Spaces

Definition (Speck, Hecher, Gnad, Fichte, and Corrêa, 2025)

Conditional probability: $$\mathbb{P}_\ell[\Pi, o] := \frac{|\,\mathsf{Plans}_\ell(\Pi, o)|}{\max(1, |\,\mathsf{Plans}_\ell(\Pi)|)}$$

Queries



Conditional Probability

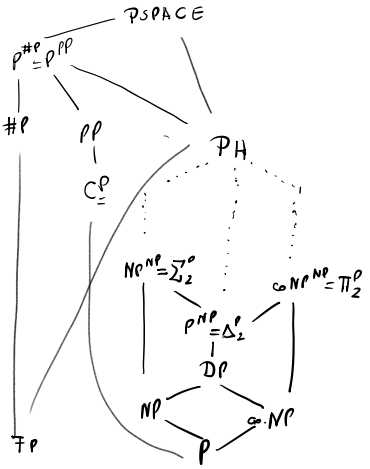
Solutions: $\mathbb{P}_\ell[P, \text{wake-up}] = 1$ $\mathbb{P}_\ell[P, \text{coffee}] = 2/3$

{{wake-up, plan-talk}, {wake-up, coffee, dl-talk}, {wake-up, coffee, asp-talk}}

Complexity Classification

(Speck, Hecher, Gnad, Fichte, and Corrêa, 2025)

Name	Task	Compl.
Poly-Bounded-Plan-Exist	$\pi \in \mathsf{Plans}_\ell(\Pi)$	NP-c
#Poly-Bounded-Plan	$\lvert \mathsf{Plans}_\ell \rvert$	#P-c
Poly-Bounded-Top-k-Exist	$\lvert \mathsf{Plans}_\ell \rvert \geq k$	PP-h
Poly-Brave-Plan-Exist	$\exists \pi \in \mathsf{Plans}_\ell(\Pi) : o \in \pi$	NP-c
Poly-Cautious-Plan-Exist	$\forall \pi \in \mathsf{Plans}_\ell(\Pi) : o \in \pi$	co-NP-c
Poly-Probabilistic-Reason	$\mathbb{P}_\ell[\Pi, Q] = p$	$\mathsf{C}^{\mathsf{P}}_{=}$-c
FacetReason	$o \in \mathcal{F}_\ell(\Pi)$	NP-c
AtLeast-k-Facets	$\lvert \mathcal{F}_\ell(\Pi) \rvert \geq k$	NP-c
AtMost-k-Facets	$\lvert \mathcal{F}_\ell(\Pi) \rvert \leq k$	co-NP-c
Exact-k-Facets	$\lvert \mathcal{F}_\ell(\Pi) \rvert = k$	D^{P}-c



Faceted Reasoning

(Speck et al., 2025; Gnad et al., 2025)

Definition

Planning task Π and an length ℓ

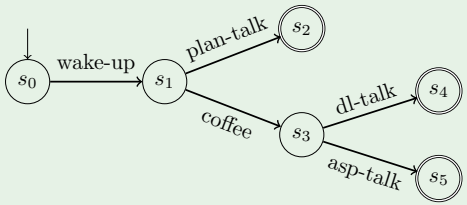
- Brave operators: used by at least one plan $\mathcal{BO}_\ell(\Pi) := \bigcup_{\pi \in \mathsf{Plans}_\ell(\Pi)} \nabla(\pi)$
- Cautious operators: used in all plans (landmark) (c.f., (Zhu and Givan, 2003)) $\mathcal{CO}_\ell(\Pi) := \bigcap_{\pi \in \mathsf{Plans}_\ell(\Pi)} \nabla(\pi)$

$\nabla(\cdot)\colon (v_1, \ldots, v_n) \rightsquigarrow \{v_1, \ldots, v_n\}$

Example (cont.)

$\mathcal{BO}_\ell(\Pi_1) = \{\text{wake-up}, \text{plan-talk}, \text{coffee}, \text{dl-talk}, \text{asp-talk}\},$

$\mathcal{CO}_\ell(\Pi_1) = \{\text{wake-up}\}.$



Faceted Reasoning (cont.)

(Speck et al., 2025; Gnad et al., 2025)

Operators that allow for flexibility / uncertainty

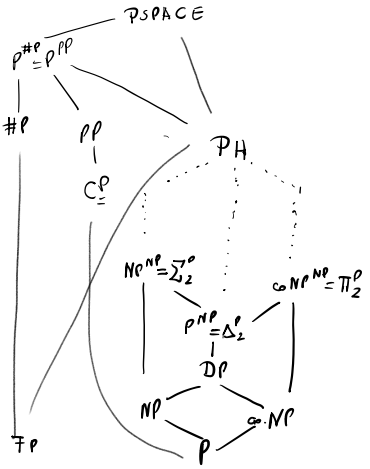
- Inclusive facets: $\mathcal{F}_\ell^+(\Pi) \coloneqq \mathcal{BO}_\ell(\Pi) \setminus \mathcal{CO}_\ell(\Pi)$ (enforce on plan)
- Significance of operator o:

$$\mathbb{S}_\ell(\Pi, o) \coloneqq \frac{|\mathcal{F}_\ell(\Pi)| - |\mathcal{F}_\ell(\Pi[o])|}{|\mathcal{F}_\ell(\Pi)|}$$

Complexity Classification

(Speck, Hecher, Gnad, Fichte, and Corrêa, 2025)

Name	Task	Compl.
Poly-Bounded-Plan-Exist	$\pi \in \mathsf{Plans}_\ell(\Pi)$	NP-c
#Poly-Bounded-Plan	$\lvert \mathsf{Plans}_\ell \rvert$	#P-c
Poly-Bounded-Top-k-Exist	$\lvert \mathsf{Plans}_\ell \rvert \geq k$	PP-h
Poly-Brave-Plan-Exist	$\exists \pi \in \mathsf{Plans}_\ell(\Pi) : o \in \pi$	NP-c
Poly-Cautious-Plan-Exist	$\forall \pi \in \mathsf{Plans}_\ell(\Pi) : o \in \pi$	co-NP-c
Poly-Probabilistic-Reason	$\mathbb{P}_\ell[\Pi, Q] = p$	$\mathsf{C}^{\mathsf{P}}_{=}$-c
FacetReason	$o \in \mathcal{F}_\ell(\Pi)$	NP-c
AtLeast-k-Facets	$\lvert \mathcal{F}_\ell(\Pi) \rvert \geq k$	NP-c
AtMost-k-Facets	$\lvert \mathcal{F}_\ell(\Pi) \rvert \leq k$	co-NP-c
Exact-k-Facets	$\lvert \mathcal{F}_\ell(\Pi) \rvert = k$	D^{P}-c

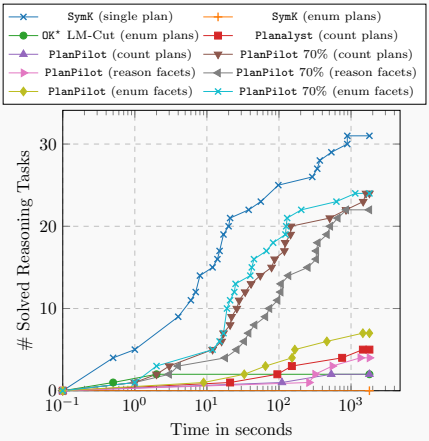


Facets & Counting: Practical Application

AI Planning

- Airbus 2025 Beluga AI explainability challenge
- Logistics application
- 48 solvable instances
- Inverted cactus plot: runtime necessary to solve or reason over a given number of instances
- x-axis runtime (log-scale)
- y-asis solved instances

(Speck, Hecher, Gnad, Fichte, and Corrêa, 2025; Gnad, Hecher, Gaggl, Rusovac, Speck, and Fichte, 2025)



Summary & Outlook

Summary

- Applications to:
  Abstract Argumentation (incl. Diversity Measures), Abductive Reasoning, Symbolic AI Planning, Answer Set Programming (ASP), Epistemic Logic Programming (ELP)
- Navigating Solution Spaces using Counting Techniques
- Pre-/inprocessing: application-specific, weighted/algebraic instances
- Certified Solving and Proof Systems (Proof Complexity)

Thanks for listening.

Bibliography

References I

Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes width parameters. Theory of Computing Systems, 64 (5):861–914, 2020. doi: 10.1007/s00224-019-09930-2. URL https://doi.org/10.1007/s00224-019-09930-2.

Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Just-in-time compilation of knowledge bases. In IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, pages 447–453, 2013.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In Chandra Sekhar Chekuri and Daniele Micciancio, editors, Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS'03), pages 340–351, Cambridge, MA, USA, October 2003. IEEE Computer Soc.

References II

Tomáš Balyo, Martin Suda, Lukáš Chrpa, Dominik Šafránek, Stephan Gocht, Filip Dvořák, Roman Barták, and G. Michael Youngblood. Planning Domain Model Acquisition from State Traces without Action Parameters. In KR'24, pages 812–822, 2024. doi: 10.24963/kr.2024/76.

Max Bannach, Giacomo Acciarini, and Dario Izzo. Reliability of constellations with inter-satellite communication. In International Astronautical Congress, 2024.

Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. Journal of Combinatorial Theory, Series A, 14(2):137–148, 1973. ISSN 0097-3165. doi: https://doi.org/10.1016/0097-3165(73)90016-2. URL https://www.sciencedirect.com/science/article/pii/0097316573900162.

References III

Olaf Beyersdorff, **Johannes K. Fichte**, Markus Hecher, Tim Hoffmann, and Kaspar Kasche. The relative strength of #sat proof systems. In Supratik Chakraborty and Jie-Hong Roland Jiang, editors, SAT'24, volume 305 of Leibniz International Proceedings in Informatics (LIPIcs), pages 5:1–5:19, Dagstuhl, Germany, 2024. Dagstuhl Publishing. ISBN 978-3-95977-334-8. doi: 10.4230/LIPIcs.SAT.2024.5.

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. Handbook of Satisfiability (2nd Edition). Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, Netherlands, February 2021. ISBN 978-1-64368-161-0. doi: 10.3233/FAIA200992.

References IV

Armin Biere, Nils Froleyks, and Wenxi Wang. CadiBack: Extracting Backbones with CaDiCaL. In Meena Mahajan and Friedrich Slivovsky, editors, 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), volume 271 of Leibniz International Proceedings in Informatics (LIPIcs), pages 3:1–3:12, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-286-0. doi: 10.4230/LIPIcs.SAT.2023.3.

Simone Bova and Stefan Szeider. Circuit treewidth, sentential decision, and query compilation. In Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '17, pages 233–246, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450341981. doi: 10.1145/3034786.3034787. URL https://doi.org/10.1145/3034786.3034787.

References V

Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. On compiling cnfs into structured deterministic dnnfs. In Marijn Heule and Sean Weaver, editors, Theory and Applications of Satisfiability Testing – SAT 2015, pages 199–214, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24318-4.

Randal E. Bryant. Numerical considerations in weighted model counting, 2025. URL https://arxiv.org/abs/2508.06264.

Florent Capelli. Knowledge compilation languages as proof systems. In SAT, pages 90–99, 2019. doi: 10.1007/978-3-030-24258-9_6.

Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In Carla E. Brodley and Peter Stone, editors, Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI'14), pages 1722–1730, Québec City, QC, Canada, 2014. The AAAI Press.

References VI

Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, Proceedings of the 3rd Annual Symposium on Theory of Computing (ACM STOC'71), pages 151–158, Shaker Heights, OH, USA, 1971. Assoc. Comput. Mach., New York. doi: 10.1145/800157.805047.

James M. Crawford and Larry D. Auton. Experimental results on the crossover point in random 3-sat. Artif. Intell., 81(1-2):31–57, 1996.

Adnan Darwiche. New advances in compiling CNF into decomposable negation normal form. In Ramón López de Mántaras and Lorenza Saitta, editors, Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, pages 328–332, 2004.

References VII

Ridhwan Dewoprabowo, Johannes K. Fichte, Piotr Jerzy Gorczyca, and Markus Hecher. A practical account into counting dung's extensions by dynamic programming. In Daniela Inclezan and Marco Maratea, editors, LPNMR'22, volume 13416 of Lecture Notes in Computer Science, pages 387–400. Springer Verlag, 2022. doi: 10.1007/978-3-031-15707-3_30.

Jeffrey M. Dudek, Vu H. N. Phan, and Moshe Y. Vardi. ADDMC: Weighted model counting with algebraic decision diagrams. In Vincent Conitzer and Fei Sha, editors, Proceedings of the 24th AAAI Conference on Artificial Intelligence AAAI'20, pages 1468–1476, 2020.

Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In AAAI, pages 4488–4494. The AAAI Press, 2017.

References VIII

Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. Theoretical Computer Science, 340(3):496–513, 2005. ISSN 0304–3975. doi: 10.1016/j.tcs.2005.03.012.

Stephen A. Fenner, Frederic Green, Steven Homer, and Randall Pruim. Determining acceptance possibility for a quantum computation is hard for the polynomial hierarchy. ECCC, TR99-003, 1999.

Johannes K. Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. ACM Journal of Experimental Algorithmics, 26, October 2021a. ISSN 1084-6654. doi: 10.1145/3459080.

Johannes K Fichte, Sarah Alice Gaggl, and Dominik Rusovac. Rushing and strolling among answer sets – navigation made easy. In AAAI'2022, 2022a.

References IX

Johannes K. Fichte, Markus Hecher, and Mohamed A. Nadeem. Plausibility reasoning via projected answer set counting - a hybrid approach. In IJCAI'22, pages 2620–2626. IJCAI Organization, 2022b. doi: 10.24963/ijcai.2022/363.

Johannes K. Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of sat. Commun. ACM, 66(6):64–72, May 2023a. ISSN 0001-0782. doi: 10.1145/3560469.

Johannes Klaus Fichte and Markus Hecher. The model counting competitions 2021-2023. CoRR, abs/2504.13842, 2025. doi: 10.48550/ARXIV.2504.13842.

References X

Johannes Klaus Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. Weighted model counting on the GPU by exploiting small treewidth. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, Proceedings of the 26th Annual European Symposium on Algorithms (ESA'18), volume 112 of Leibniz International Proceedings in Informatics (LIPIcs), pages 28:1–28:16. Dagstuhl Publishing, 2018. ISBN 978-3-95977-081-1. doi: 10.4230/LIPIcs.ESA.2018.28.

Johannes Klaus Fichte, Markus Hecher, and Markus Zisser. An improved gpu-based SAT model counter. In Thomas Schiex and Simon de Givry, editors, Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings, volume 11802 of Lecture Notes in Computer Science, pages 491–509. Springer, 2019. doi: 10.1007/978-3-030-30048-7_29.

References XI

Johannes Klaus Fichte, Markus Hecher, and Valentin Roland. Parallel model counting with CUDA: algorithm engineering for efficient hardware utilization. In Laurent D. Michel, editor, 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021, volume 210 of LIPIcs, pages 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021b. doi: 10.4230/LIPICS.CP.2021.24.

Johannes Klaus Fichte, Markus Hecher, Michael Morak, Patrick Thier, and Stefan Woltran. Solving projected model counting by utilizing treewidth and its limits. Artif. Intell., 314:103810, 2023b. doi: 10.1016/J.ARTINT.2022.103810.

John Gill. Computational complexity of probabilistic turing machines. SIAM Journal on Computing, 6(4):675–695, 1977.

Guillaume Girol, Guilhem Lacombe, and Sébastien Bardin. Quantitative robustness for vulnerability assessment. In Proc. of PLDI, 2024. doi: 10.1145/3656407.

References XII

Daniel Gnad, Markus Hecher, Sarah Gaggl, Dominik Rusovac, David Jakob Speck, and Johannes K. Fichte. Interactive exploration of plan spaces. In Under Review, 2025.

Tuukka Korhonen and Matti Järvisalo. Integrating Tree Decompositions into Decision Heuristics of Propositional Model Counters. In Laurent D. Michel, editor, Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming (CP'21), volume 210 of Leibniz International Proceedings in Informatics (LIPIcs), pages 8:1–8:11, Virtual, 2021. Dagstuhl Publishing. ISBN 978-3-95977-211-2. doi: 10.4230/LIPIcs.CP.2021.8.

Jean-Marie Lagniez and Pierre Marquis. Preprocessing for propositional model counting. In AAAI'14, pages 2688–2694, 2014.

References XIII

Jean-Marie Lagniez and Pierre Marquis. An improved decision-DDNF compiler. In Carles Sierra, editor, Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17), pages 667–673, Melbourne, VIC, Australia, 2017. The AAAI Press.

Jean-Marie Lagniez and Pierre Marquis. About caching in D4 2.0. In 2nd International Workshop on Counting and Sampling (MCW), 2021. Working paper available on HAL.

Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Improving model counting by leveraging definability. In Proc. of IJCAI, pages 751–757, 2016.

Leonid Levin. Universal sequential search problems. Problems of Information Transmission, 9(3):265—266, 1973.

References XIV

Jingyi Mei, Tim Coopmans, Marcello Bonsangue, and Alfons Laarman. Equivalence checking of quantum circuits by model counting. In Proc. of IJCAR, pages 401–421, 2024. ISBN 978-3-031-63501-4.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In Proceedings of the 38th Design Automation Conference, DAC 2001, pages 530–535. ACM, 2001.

Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In Qiang Yang and Michael Wooldridge, editors, Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15), pages 3141–3148. The AAAI Press, 2015. ISBN 978-1-57735-738-4.

Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of facets (and some facets of complexity). pages 255–260.

References XV

Neil Robertson and P.D. Seymour. Graph minors. i. excluding a forest. Journal of Combinatorial Theory, Series B, 35(1):39–61, 1983. ISSN 0095-8956. doi: http://dx.doi.org/10.1016/0095-8956(83)90079-5. URL http://www.sciencedirect.com/science/article/pii/0095895683900795.

Marko Samer and Stefan Szeider. Algorithms for propositional model counting. J. Discrete Algorithms, 8(1):50–64, 2010. doi: 10.1016/J.JDA.2009.06.002. URL https://doi.org/10.1016/j.jda.2009.06.002.

Tian Sang, Fahiem Bacchus, Paul Beame, Henry Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In Holger H. Hoos and David G. Mitchell, editors, Online Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), pages 1–9, Vancouver, BC, Canada, 2004.

References XVI

Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In Proc. of IJCAI, pages 1169–1176, 2019a.

Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, pages 1169–1176. ijcai.org, 2019b.

Mate Soos and Kuldeep S. Meel. Arjun: An efficient independent support computation technique and its applications to counting and sampling. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392174. doi: 10.1145/3508352.3549406. URL https://doi.org/10.1145/3508352.3549406.

References XVII

David Speck, Markus Hecher, Daniel Gnad, Johannes Klaus Fichte, and Augusto B. Corrêa. Counting and reasoning with plans. In Toby Walsh, Julie Shah, and Zico Kolter, editors, AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA, pages 26688–26696. AAAI Press, 2025. doi: 10.1609/AAAI.V39I25.34871.

Samuel Teuber and Alexander Weigl. Quantifying software reliability via model-counting. In Alessandro Abate and Andrea Marin, editors, Quantitative Evaluation of Systems, pages 59–79, Cham, 2021. Springer International Publishing. ISBN 978-3-030-85172-9.

References XVIII

Marc Thurley. sharpSAT – counting models with advanced component caching and implicit BCP. In Armin Biere and Carla P. Gomes, editors, Proceedings of the 9th International Conference Theory and Applications of Satisfiability Testing (SAT'06), pages 424–429, Seattle, WA, USA, 2006. Springer Verlag. ISBN 978-3-540-37207-3. doi: 10.1007/11814948_38.

Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. SIAM J. Comput., 20 (5):865–877, October 1991. ISSN 0097-5397. doi: 10.1137/0220053.

Leslie G. Valiant. The complexity of computing the permanent. Theoretical Computer Science, 8(2):189–201, 1979. ISSN 0304-3975. doi: 10.1016/0304-3975(79)90044-6.

Lin Zhu and Robert Givan. Landmark extraction via planning graph propagation. In ICAPS 2003 Doctoral Consortium, pages 156–160, 2003.

Problem of Interest (cont.)

#∃SAT-Problem (Projected Model Counting)

Given: Propositional formula F and a set $P \subseteq \mathrm{var}(F)$ of projection variables.
Question: Output the number of models projected to the projection variables.
$\mathrm{pmc}(F) := |\{ M \cap P : M \in \mathrm{Mod}(F)|$[a]

[a] Intuition: "auxiliary" variables important for SAT; ignore when counting

Example (Projected Model Count)

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

$$\mathrm{pmc}(F, P) = |\{\underset{\cap P}{\{b\}}, \underset{\cap P}{\{a, b\}}, \underset{\cap P}{\{b, c\}}, \underset{\cap P}{\{a, b, c\}}, \underset{\cap P}{\{b, c, x\}}, \underset{\cap P}{\{a, b, c, x\}}, \underset{\cap P}{\{b, y\}}, \underset{\cap P}{\{a, b, y\}}\}|$$

- Relevant for encodings that introduce auxiliary variables

Problem of Interest (cont.)

#∃SAT-Problem (Projected Model Counting)

Given: Propositional formula F and a set $P \subseteq \mathsf{var}(F)$ of projection variables.
Question: Output the number of models projected to the projection variables.
$\mathsf{pmc}(F) := |\{ M \cap P : M \in \mathsf{Mod}(F)|$[a]

[a] Intuition: "auxiliary" variables important for SAT; ignore when counting

Example

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

$$\mathsf{pmc}(F, P) = |\{\emptyset, \emptyset, \emptyset, \emptyset, \{x\}, \{x\}, \{y\}, \{y\}\}| = 3$$

- Relevant for encodings that introduce auxiliary variables

Backup Slides

Certifying the Results of Model Counters

Proofs

- Solvers make mistakes, change constantly
- Solvers cannot be verified entirely $\Rightarrow$ trace output and certify the results
- MICE FichteHecherRoland21
  - Define proof system based on basic search space splitting techniques
  - Reuse existing SAT proof techniques (DRAT)
  - Building blocks: components/assumptions
    exactly one model, compose, join, extend rules
- Equivalence Checking
  - kcps (knowledge compilation proof system) (Capelli, 2019)
  - CPOG (certified partitioned-operation graphs) (Bryant, 2025)

Compare relative strength (Beyersdorff et al., 2024)

Modern Solving Techniques (cont.)

Solvers Model Counting Competition 2021–23

- Caching:
  - Cachet: store previously seen solutions (Sang et al., 2004)
  - SharpSAT: store solutions more compactly (Thurley, 2006)
  - Ganak: reduce the space for cache by a hash function (Sharma et al., 2019a)
    (small probability of returning wrong count)
- Branching Heuristics
  - SharpSAT-TD (Korhonen and Järvisalo, 2021)
    $\text{score}(x) = \text{freq}(x) + \text{act}(x) - C \cdot d(x)$,
    freq ... frequency, act... activity
    $d(x)$... distance in the tree decomposition from the root node to a closest node
    $C = 100 exp(n/w)/n$, where w is the width of the tree
    decomposition and n is the number of variables

Modern Solving Techniques (cont.)

- Preprocessing
  - pmc (Lagniez and Marquis, 2014)
    - complete vification
      (c and a literal l in c check if $\neg(c \setminus \{\ell\})$ implies UNSAT via unit propagation, and if yes we strenghten c by removing ℓ)
  - B+E (Lagniez et al., 2016)
    - definability between variables + reduce by eliminating variables which have been identified as outputs of some gates
  - SharpSAT-TD (Korhonen and Järvisalo, 2021)
    - fast vification (propagation-based vivification)
    - complete vivification
    - sparsification (remove clauses that are implied by other clauses, redundancy check with SAT-solver),
    - equivalence merging (merge two variables if they are equivalent and they are adjacent in the primal graph)
    - B+E (identify bounded variables)
  - Arjun: + minimal independent sets and use this for projections (Soos and Meel, 2022)
  - CaDiBack: backbone analyzing (Biere et al., 2023)

Modern Solving Techniques (cont.)

- Knowledge-compilation
  - d4 (glucose): dynamic decompositions based on hypergraph partitioning
- Dynamic-Programming
  - DPMC: algebraic decision diagrams
- ...

Modern Solving Techniques: Component Caching

The Naive Approach

Search

The Solution: On-the-fly Caching

- Maintain **Cache** (a hash map) storing pairs of $\langle \texttt{CNF}, n \rangle$.
- Decision: check if the current remaining CNF sub-problem is already in the map.
- **If yes:** We link to the existing n.
- **If no:** We compute, and store.

The Implementation Catch

- Testing if two formulas are logically equivalent is coNP-complete. We cannot do this at every step!
- **In practice:** We check for **Syntactic Identity** (are the clauses exactly the same, up to ordering?). This is fast via hashing.

Step 1: Representing a Single Clause

Since a component is a set of clauses, we first need a way to represent individual clauses in the cache key.

1. Literal Representation

The clause is explicitly stored as a sorted list of literals.

- **Format:** Integers (Variable IDs).
- **Sign:** Positive ID for x_i, negative for $\neg x_i$.
- **Terminator:** Ends with 0.

Ex: $(x_1 \vee \neg x_3) \rightarrow [1, -3, 0]$

2. Index Representation

The clause is represented by its unique ID from the original CNF.

- **Format:** A single integer (Index) and a list of the variables indices.
- **Pros:** Very compact (low memory).
- **Cons:** Less precise.

Ex: $\#42{:}(x_1 \vee x_3) \rightarrow [42][1, 3]$

Running Example: Setup

Let us consider a specific CNF F and a partial assignment.

Original CNF F

Indices:

1. $(x_1 \vee x_2 \vee x_4)$
2. $(x_2 \vee x_3 \vee x_4)$
3. $(x_3 \vee x_4 \vee x_5)$
4. $(x_1 \vee \neg x_6)$

Current State (φ)

Decision: Set $x_2 = \top$. **BCP (Unit Propagation):**

- Clause 1 $(x_1 \vee \top \vee x_4)$ is Satisfied (Removed).
- Clause 2 $(\top \vee x_3 \vee x_4)$ is Satisfied (Removed).

Remaining Component φ:

$$(x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee \neg x_6)$$

Active Variables: $\{1, 3, 4, 5, 6\}$

Example 1: "All Clauses" Scheme

Stores the variable set plus **every** active clause.

Remaining Clauses: $\underbrace{(x_3 \vee x_4 \vee x_5)}_{\text{Clause 3}} \wedge \underbrace{(x_1 \vee \neg x_6)}_{\text{Clause 4}}$

Representations

- **Literal Representation:** Explicitly list literals for both clauses.

$$r(\varphi) = \big([\underbrace{3, 4, 5, 0}_{C_3}, \underbrace{1, -6, 0}_{C_4}]\big)$$

- **Index Representation:** List the IDs of the active clauses plus variables.

$$r(\varphi) = \big([1, 3, 4, 5, 6], \quad [3, 4]\big)$$

Example 2: "Non-Binary" Scheme (sharpSAT)

Stores variables plus only **large** clauses (Size > 2).

$$\text{Active Clauses: } \underbrace{(x_3 \vee x_4 \vee x_5)}_{\text{Size 3 (Keep)}} \wedge \underbrace{(x_1 \vee \neg x_6)}_{\text{Size 2 (Drop)}}$$

Representations

- **Literal Representation:** Only the ternary clause is stored explicitly.

$$r(\varphi) = \big([1,3,4,5,6], \quad [3,4,5,0]\big)$$

- **Index Representation:** Only the index of the ternary clause.

$$r(\varphi) = \big([1,3,4,5,6], \quad [3]\big)$$

Example 3: "Not Touched" Scheme (D4)

Stores variables plus only **modified** clauses.

- $C_3 : (x_3 \vee x_4 \vee x_5)$. Original: Same. **Status: Untouched.**
- $C_4 : (x_1 \vee \neg x_6)$. Original: Same. **Status: Untouched.**

Since neither clause was modified by the assignment $x_2 = \top$ (they didn't contain x_2), they are implicit.

Representations

- **Literal & Index Representation:** The clause list is empty! The component is identified solely by its variables.

$$r(\varphi) = \big([1, 3, 4, 5, 6], \quad [\emptyset]\big)$$

Note: If a clause had been shortened (e.g., $(x_3 \vee x_4)$ from an original $(x_3 \vee x_4 \vee \neg x_2)$), it would be included here.

Towards Abstractions for DP

(Fichte et al., 2023b)

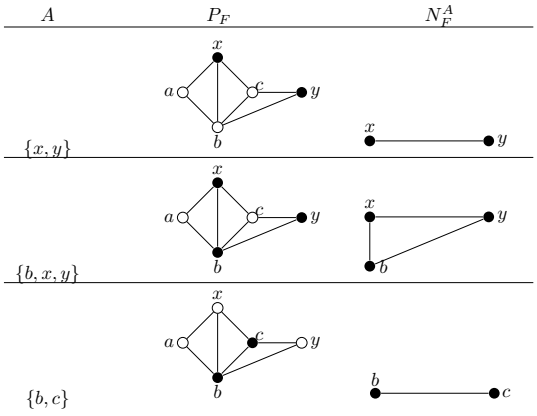
- Goal: Use SAT solvers to for solving hard subproblems
- Idea: Try to use abstractions of P_F whose treewidth k is small

$\rightsquigarrow$ Decompose only "parts" of F restricted to a set $A \subseteq \mathrm{var}(F)$

Nested primal graph $N_F^A = (A, E)$

- Vertices are a given set A of abstraction variables
- Vertices $u, v \in A$ are connected, i.e., $\{u, v\} \in E$, iff there is a path from u to v in P_F using no other vertex of A

Primal graph P_F vs. Nested primal graph N_F^A



- Works for PMC
- $\Rightarrow$ nestHDB: width up to 99
- Other symbolic AI formalisms
  (Fichte et al., 2022b)
  (Dewoprabowo, Fichte, Gorczyca, and Hecher, 2022)

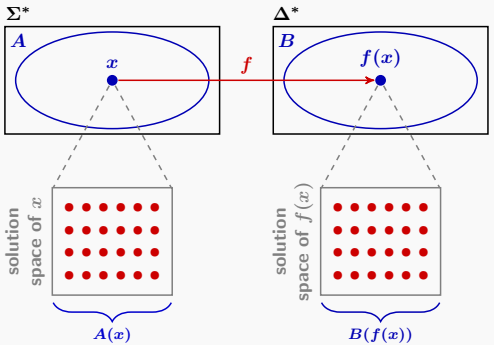
Computational Complexity

Background

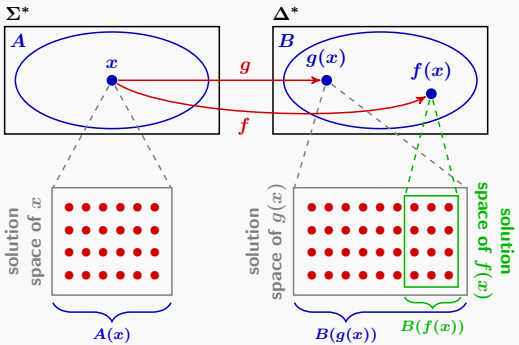
- #P: counting counter part of NP, i.e., problems solved by counting accepting paths of a nondeterministic TM (Valiant, 1979)
- D^{P}: (independent) combination of an NP and a co-NP problem, i.e., $\mathsf{D}^{\mathsf{P}} := \{ L_1 \cap L_2 : L_1 \in \mathsf{NP}, L_2 \in \mathsf{co\text{-}NP} \}$ (Papadimitriou and Yannakakis)
- PP: decision problems solved by nondeterministic TM, s.t. positive instances are those where at least $1/2$ of the machine's paths are accepting (Gill, 1977)
- $\mathsf{C}_{=}^{\mathsf{P}}$: decision problems solved by nondeterministic TM where positive instances are those with the same number of accepting and rejecting paths (Fenner et al., 1999)

Computational Complexity

Reductions



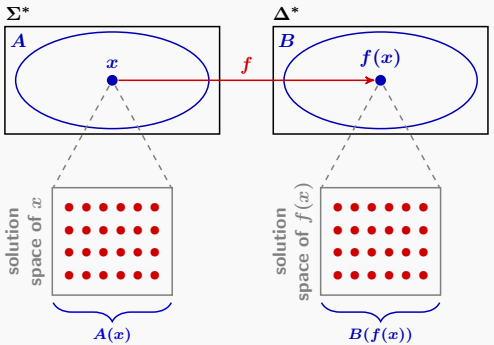
parsimonious reductions maintain cardinalities of the solution spaces, so $|A(x)| = |B(f(x))|$



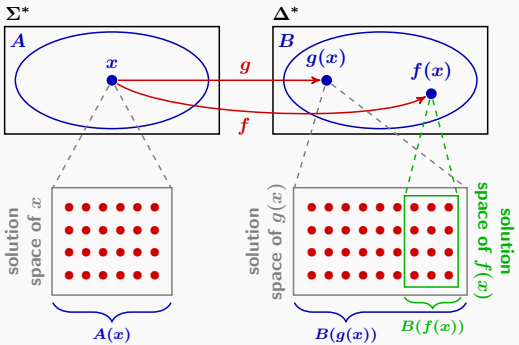
subtractive reductions recalculate cardinalities: $|A(x)| = |B(g(x))| - |B(f(x))|$

Computational Complexity

Reductions



parsimonious reductions maintain cardinalities of the solution spaces, so $|A(x)| = |B(f(x))|$



subtractive reductions recalculate cardinalities: $|A(x)| = |B(g(x))| - |B(f(x))|$

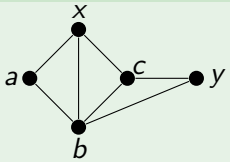
Dynamic Programming for PMC

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

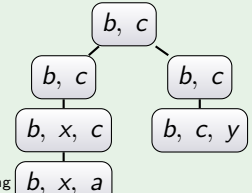
$$\begin{aligned} \mathsf{Mod}(F) &= \{ \quad \{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \\ &\qquad \{b, c, x\}, \{a, b, c, x\}, \\ &\qquad \{b, y\}, \{a, b, y\}\} \\ \mathsf{PMC}(F, P) &= \quad |\{\emptyset, \{x\}, \{y\}\}| = 3 \end{aligned}$$

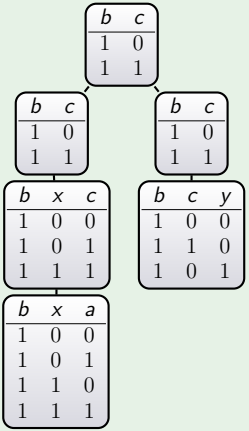
Dynamic Programming for PMC

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



1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$





b	c
1	0
1	1

b	c
1	0
1	1

b	c
1	0
1	1

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

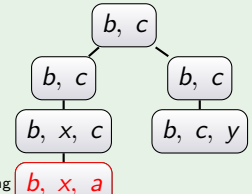
b	c	y
1	0	0
1	1	0
1	0	1

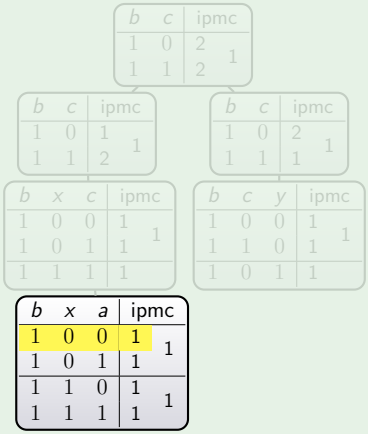
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1

Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

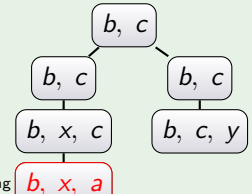


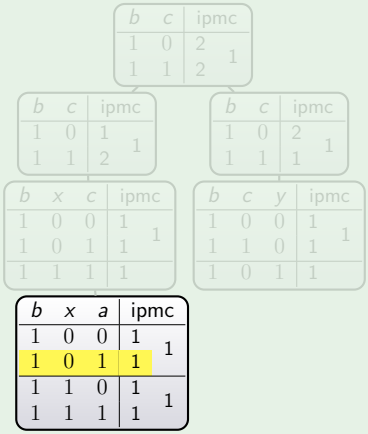


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

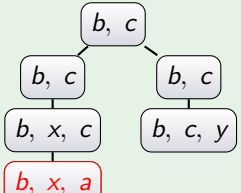




Dynamic Programming for PMC

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

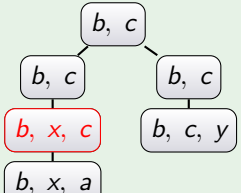
1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

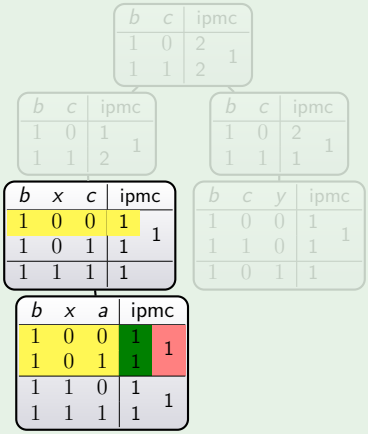


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

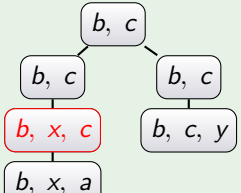


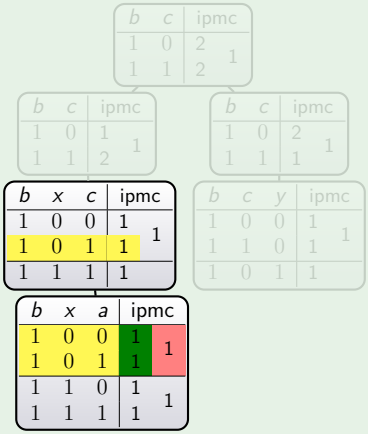


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

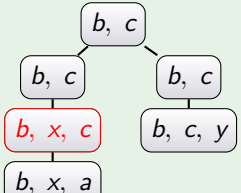


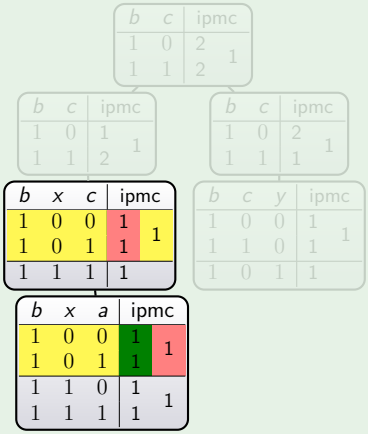


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

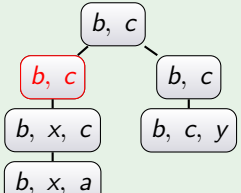


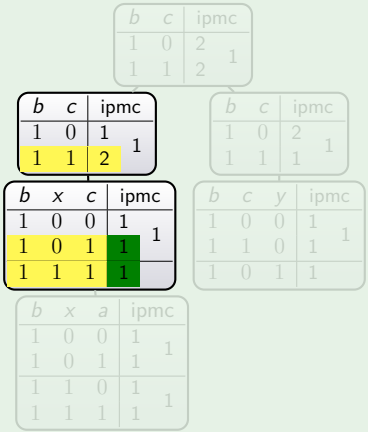


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

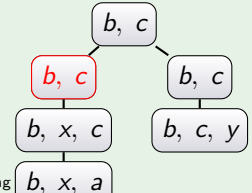


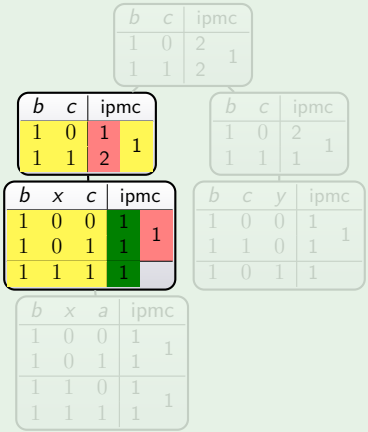


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

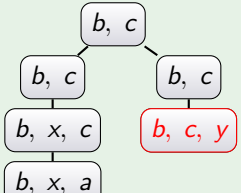


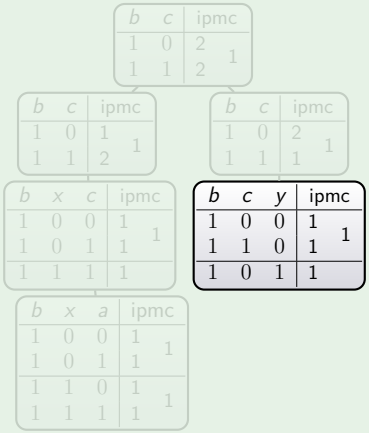


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

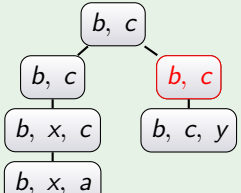


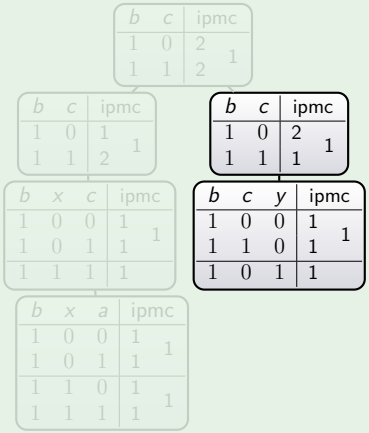


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

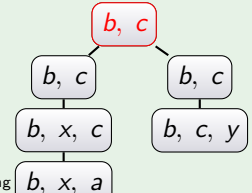


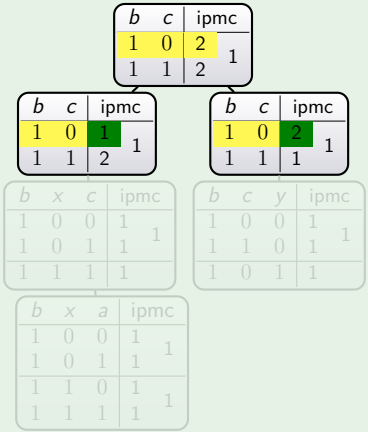


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

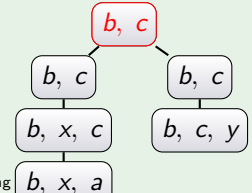


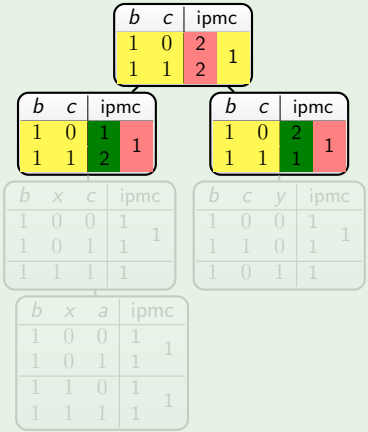


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$

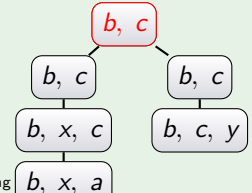


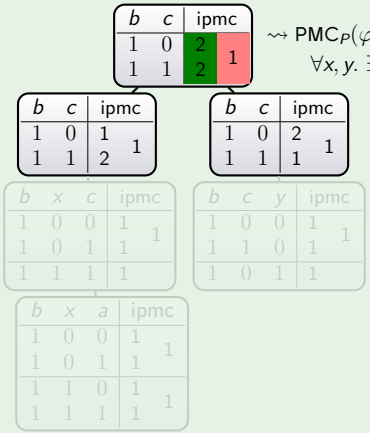


Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$





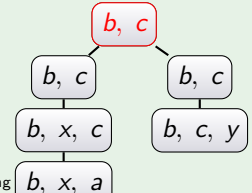
$\rightsquigarrow \mathsf{PMC}_P(\varphi) = 2 + 2 - 1 = 3$

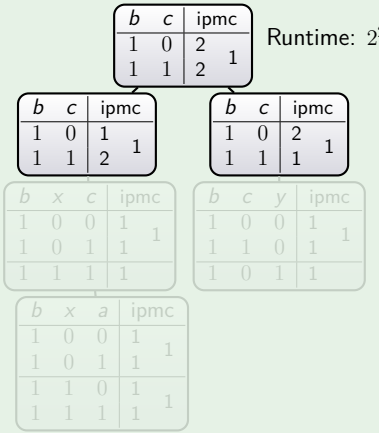
$\forall x, y.\ \exists a, b, c.\ \varphi$ invalid

Dynamic Programming for PMC

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

1. Local algorithm $\mathcal{S}$
2. Purge non-solutions
3. Solve PMC via $\mathcal{P}$





Runtime: $2^{2^{O(tw)}} \cdot \text{poly}(|\varphi|)$

Bayesian Inference as Counting Problem

What possible combinations can we obtain?

Conjecture	Possible Paths for $\langle$●●●$\rangle$
[●●●●]	0
[○●●●]	9
[○○●●]	8
[○○○●]	3
[○○○○]	0
Total:	20

Bayesian Inference as Counting Problem

What possible combinations can we obtain?

Conjecture	Possible Paths for ⟨●●●⟩
[●●●●]	0
[●●●●]	9
[●●●●]	8
[●●●●]	3
[●●●●]	0
Total:	20

Plausibility: relationship between actual and possible occurrence

Bayesian Inference as Counting Problem

What possible combinations can we obtain?

Conjecture	Possible Paths for $\langle$●●●$\rangle$
[●●●●]	0
[●●●●]	9
[●●●●]	8
[●●●●]	3
[●●●●]	0
Total:	20

Plausibility [●●●●] "=" $\frac{9}{0+9+8+3+0} = \frac{9}{20}$

Bayesian Inference as Counting Problem

What possible combinations can we obtain?

Conjecture	Possible Paths for $\langle$●●●$\rangle$	Plausibility
[●●●●]	0	$0/20 = 0$
[●●●●]	9	$9/20 = 0.45$
[●●●●]	8	$8/20 = 0.40$
[●●●●]	3	$3/20 = 0.15$
[●●●●]	0	$0/20 = 0$
Total:	20	1

Bayesian Inference as Counting Problem

What possible combinations can we obtain?

Conjecture	Possible Paths for ⟨●●●⟩	Plausibility
[●●●●]	0	$0/20 = 0$
[●●●●]	9	$9/20 = 0.45$
[●●●●]	8	$8/20 = 0.40$
[●●●●]	3	$3/20 = 0.15$
[●●●●]	0	$0/20 = 0$
Total:	20	1

Plausibility: relationship between actual and possible occurrence

Bayesian Inference as Counting Problem

What possible combinations can we obtain?

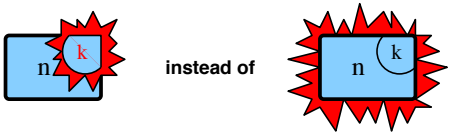
Conjecture	Possible Paths for $\langle$●●●$\rangle$	Plausibility
[●●●●]	0	$0/20 = 0$
[●●●●]	9	$9/20 = 0.45$
[●●●●]	8	$8/20 = 0.40$
[●●●●]	3	$3/20 = 0.15$
[●●●●]	0	$0/20 = 0$
Total:	20	1

Plausibility [●●●●] "=" $\frac{9}{0+9+8+3+0} = \frac{9}{20}$

Parameterized Complexity

Treewidth k (Bertelè and Brioschi, 1973; Robertson and Seymour, 1983)

- renders large variety of NP-hard problems fixed-parameter tractable (FPT)

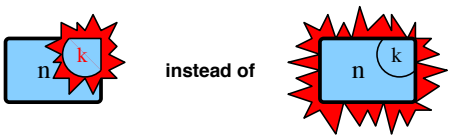


$f(k) \cdot poly(n)$ vs. 2^n

Parameterized Complexity

Treewidth k (Bertelè and Brioschi, 1973; Robertson and Seymour, 1983)

- renders large variety of NP-hard problems fixed-parameter tractable (FPT)



$f(k) \cdot poly(n)$ vs. 2^n

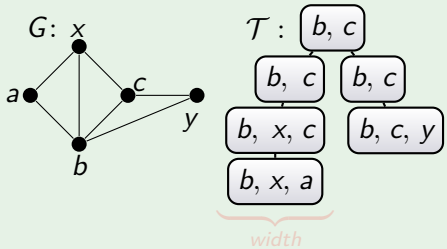
Practical Perspective

- Precise $f(k)$? Can we do better?

$\Rightarrow$ Parameterized algorithmics and engineering

Tree Decompositions (TDs)

Tree Decomposition $\mathcal{T}$ of G



Definition

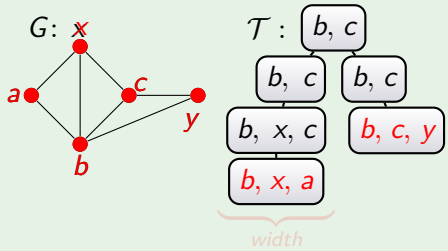
(Robertson and Seymour, 1983)

A tree decomposition is a tree of bags obtained from a graph s.t.

1. Every vertex must occur in some bag
2. Every edge must be covered by some bag
3. Connected: Tree "restricted" to any vertex must be connected

Tree Decompositions (TDs)

Tree Decomposition $\mathcal{T}$ of G



Definition

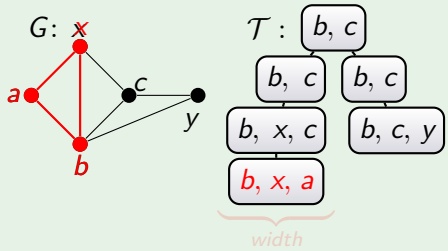
(Robertson and Seymour, 1983)

A tree decomposition is a tree of bags obtained from a graph s.t.

1. Every vertex must occur in some bag
2. Every edge must be covered by some bag
3. Connected: Tree "restricted" to any vertex must be connected

Tree Decompositions (TDs)

Tree Decomposition $\mathcal{T}$ of G



Definition

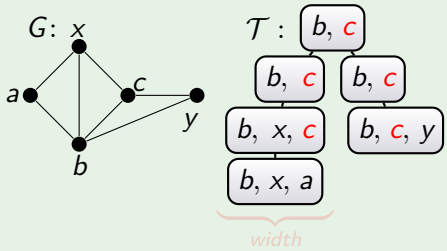
(Robertson and Seymour, 1983)

A tree decomposition is a tree of bags obtained from a graph s.t.

1. Every vertex must occur in some bag
2. Every edge must be covered by some bag
3. Connected: Tree "restricted" to any vertex must be connected

Tree Decompositions (TDs)

Tree Decomposition $\mathcal{T}$ of G



Definition

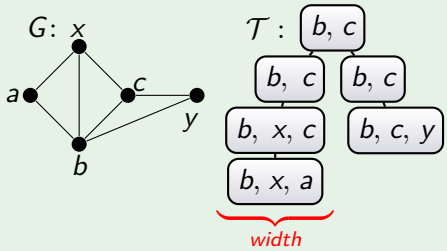
(Robertson and Seymour, 1983)

A tree decomposition is a tree of bags obtained from a graph s.t.

1. Every vertex must occur in some bag
2. Every edge must be covered by some bag
3. Connected: Tree "restricted" to any vertex must be connected

Tree Decompositions (TDs)

Tree Decomposition $\mathcal{T}$ of G



Definition

(Robertson and Seymour, 1983)

A tree decomposition is a tree of bags obtained from a graph s.t.

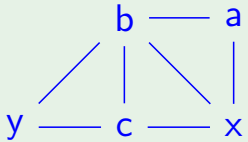
1. Every vertex must occur in some bag
2. Every edge must be covered by some bag
3. Connected: Tree "restricted" to any vertex must be connected

Solving MC

(Samer and Szeider, 2010)

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

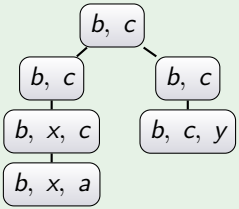


Solving MC

(Samer and Szeider, 2010)

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

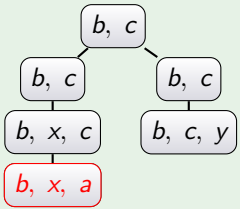


Solving MC

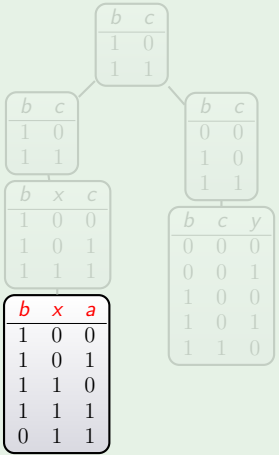
(Samer and Szeider, 2010)

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows



"Local formula" F_t clauses whose variables are contained in the bag (colored in red above)

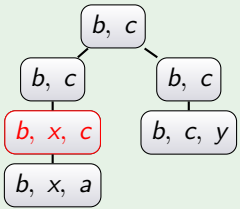


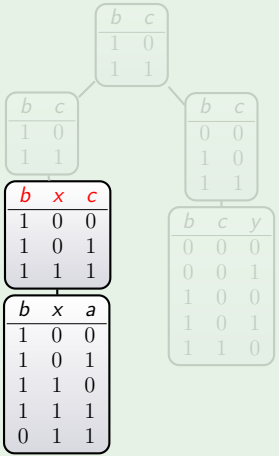
Solving MC

(Samer and Szeider, 2010)

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows



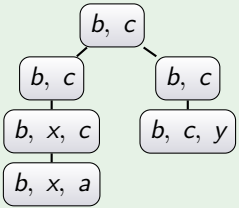


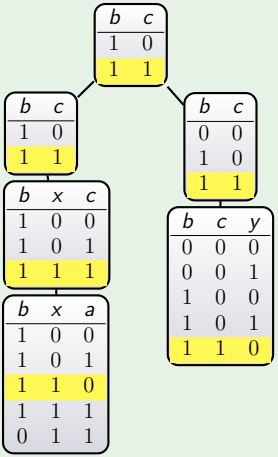
Solving MC

(Samer and Szeider, 2010)

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows





Towards projection

Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$

Towards projection

Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$

Towards projection

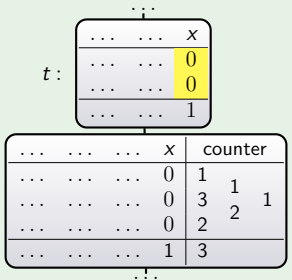
Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$



Towards projection

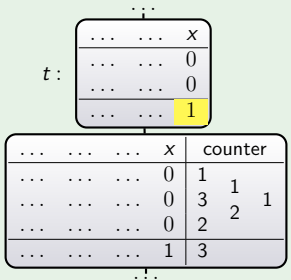
Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$



Towards projection

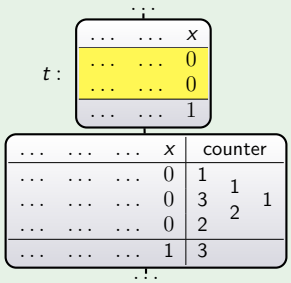
Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$



Towards projection

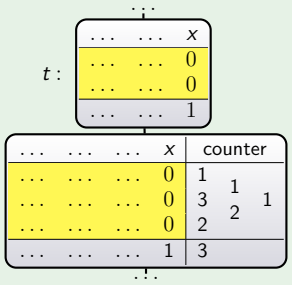
Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$



Towards projection

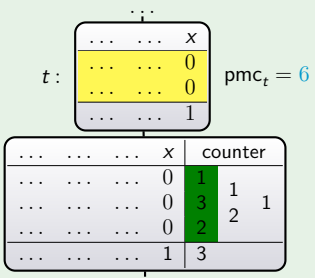
Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$



Towards projection

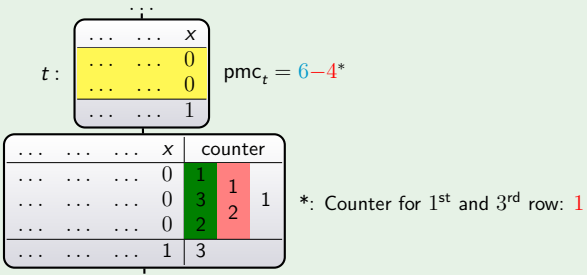
Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$



Towards projection

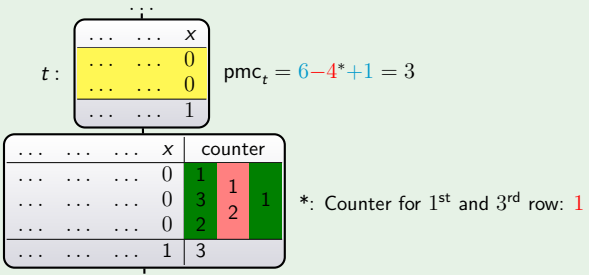
Ingredients for given PMC instance (F, P)

- Equivalence classes (partitioning) of table rows

  For rows u, v of a table: $u \equiv_P v \iff I(u) \cap P = I(v) \cap P$

- Map subsets of resulting "row classes" to a counter
- Maintain counters using the principle of inclusion and exclusion

Example: Computing pmc_t given $P = \{x, y, z\}$



Towards projection

Principle of Inclusion-Exclusion (PIE) for family $\mathcal{X}$ of finite sets X_i

$$\begin{aligned} |\bigcup_{X_i \in \mathcal{X}} X_i| &= |X_1| + |X_2| + \ldots - |X_1 \cap X_2| + \ldots + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i| \\ &= \Sigma_{I \subsetneq \{1,\ldots,|\mathcal{X}|\}} (-1)^{|I|-1} \cdot |\bigcap_{i \in I} X_i| \quad + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i| \end{aligned}$$

Towards projection

Principle of Inclusion-Exclusion (PIE) for family $\mathcal{X}$ of finite sets X_i

$$
\begin{aligned}
|\bigcup_{X_i \in \mathcal{X}} X_i| &= |X_1| + |X_2| + \ldots - |X_1 \cap X_2| + \ldots + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i| \\
&= \Sigma_{I \subsetneq \{1,\ldots,|\mathcal{X}|\}} (-1)^{|I|-1} \cdot |\bigcap_{i \in I} X_i| \qquad + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i|
\end{aligned}
$$

Towards projection

Principle of Inclusion-Exclusion (PIE) for family $\mathcal{X}$ of finite sets X_i

$$\underbrace{|\bigcup_{X_i \in \mathcal{X}} X_i|}_{\mathsf{pmc}_t} = |X_1| + |X_2| + \ldots - |X_1 \cap X_2| + \ldots + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i|$$

$$= \Sigma_{I \subsetneq \{1,\ldots,|\mathcal{X}|\}} (-1)^{|I|-1} \cdot |\bigcap_{i \in I} X_i| \qquad + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i|$$

Towards projection

Principle of Inclusion-Exclusion (PIE) for family $\mathcal{X}$ of finite sets X_i

$$\underbrace{|\bigcup_{X_i \in \mathcal{X}} X_i|}_{\text{pmc}_t} = |X_1| + |X_2| + \ldots - |X_1 \cap X_2| + \ldots + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i|$$

$$= \Sigma_{I \subsetneq \{1,\ldots,|\mathcal{X}|\}} (-1)^{|I|-1} \cdot |\bigcap_{i \in I} X_i| \quad + (-1)^{|\mathcal{X}|-1} \cdot \underbrace{|\bigcap_{X_i \in \mathcal{X}} X_i|}_{\text{ipmc}_t}$$

Towards projection

Principle of Inclusion-Exclusion (PIE) for family $\mathcal{X}$ of finite sets X_i

$$\underbrace{|\bigcup_{X_i \in \mathcal{X}} X_i|}_{\text{pmc}_t} = |X_1| + |X_2| + \ldots - |X_1 \cap X_2| + \ldots + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i|$$

$$= \Sigma_{I \subsetneq \{1,\ldots,|\mathcal{X}|\}} (-1)^{|I|-1} \cdot |\bigcap_{i \in I} X_i| \quad + (-1)^{|\mathcal{X}|-1} \cdot \underbrace{|\bigcap_{X_i \in \mathcal{X}} X_i|}_{\text{ipmc}_t}$$

Towards projection

Principle of Inclusion-Exclusion (PIE) for family $\mathcal{X}$ of finite sets X_i

$$\underbrace{|\bigcup_{X_i \in \mathcal{X}} X_i|}_{\text{pmc}_t} = |X_1| + |X_2| + \ldots - |X_1 \cap X_2| + \ldots + (-1)^{|\mathcal{X}|-1} \cdot |\bigcap_{X_i \in \mathcal{X}} X_i|$$

$$= \Sigma_{I \subsetneq \{1,\ldots,|\mathcal{X}|\}} (-1)^{|I|-1} \cdot |\bigcap_{i \in I} X_i| \quad + (-1)^{|\mathcal{X}|-1} \cdot \underbrace{|\bigcap_{X_i \in \mathcal{X}} X_i|}_{\text{ipmc}_t}$$

Local algorithm $\mathcal{P}$ in more details:

- Only store ("counter") ipmc_t-values
- pmc_t-values are used intermediately
- Generalization of PIE that – instead of cardinalities – relies on stored ipmc_t-values

Paralleling Dynamic Programming

How to parallelize DP?

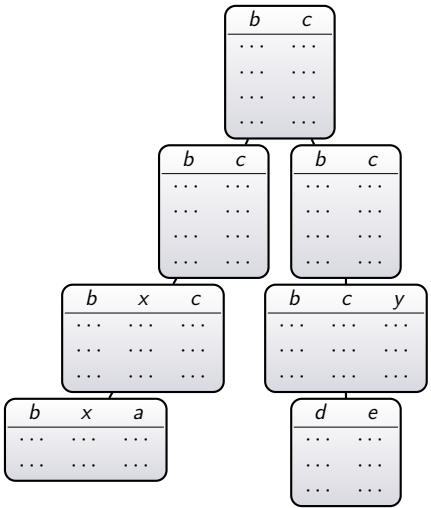
1. Compute tables for multiple nodes in parallel

⇒ 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 for PMC

Implementation nestHDB

Framework based on dpdb using database systems [FHecherThierWoltran'21]

- Why databases?
  1. Efficient table manipulation operations (selection, projection and joins)
  2. Convenient representation of table algorithms
- written in Python3, multi-core support, uses PostgreSQL 12 on ramdisk
- open-source: https://github.com/hmarkus/dp_on_dbs/tree/nesthdb
- relies on SAT solver picosat, MC solver MC and PM solver projMC for search-intense subproblems $\rightsquigarrow$ Hybrid Solving
- "Good" nested primal graphs (abstractions) are approximated via a logic program
  - prefers abstractions that are large, but not dense, i.e., whose treewidth is small
  - uses built-in optimization to get the best obtained abstraction after 35 seconds

Implementation nestHDB

Framework based on dpdb using database systems [FHecherThierWoltran'21]

- Why databases?
  1. Efficient table manipulation operations (selection, projection and joins)
  2. Convenient representation of table algorithms
- written in Python3, multi-core support, uses PostgreSQL 12 on ramdisk
- open-source: https://github.com/hmarkus/dp_on_dbs/tree/nesthdb
- relies on SAT solver picosat, MC solver MC and PM solver projMC for search-intense subproblems $\rightsquigarrow$ Hybrid Solving
- "Good" nested primal graphs (abstractions) are approximated via a logic program
  - prefers abstractions that are large, but not dense, i.e., whose treewidth is small
  - uses built-in optimization to get the best obtained abstraction after 35 seconds

Implementation nestHDB

Framework based on dpdb using database systems [FHecherThierWoltran'21]

- Why databases?
  1. Efficient table manipulation operations (selection, projection and joins)
  2. Convenient representation of table algorithms
- written in Python3, multi-core support, uses PostgreSQL 12 on ramdisk
- open-source: https://github.com/hmarkus/dp_on_dbs/tree/nesthdb
- relies on SAT solver picosat, MC solver MC and PM solver projMC for search-intense subproblems $\rightsquigarrow$ Hybrid Solving
- "Good" nested primal graphs (abstractions) are approximated via a logic program
  - prefers abstractions that are large, but not dense, i.e., whose treewidth is small
  - uses built-in optimization to get the best obtained abstraction after 35 seconds

Implementation nestHDB

Framework based on dpdb using database systems [FHecherThierWoltran'21]

- Why databases?
  1. Efficient table manipulation operations (selection, projection and joins)
  2. Convenient representation of table algorithms
- written in Python3, multi-core support, uses PostgreSQL 12 on ramdisk
- open-source: https://github.com/hmarkus/dp_on_dbs/tree/nesthdb
- relies on SAT solver picosat, MC solver MC and PM solver projMC for search-intense subproblems $\rightsquigarrow$ Hybrid Solving
- "Good" nested primal graphs (abstractions) are approximated via a logic program
  - prefers abstractions that are large, but not dense, i.e., whose treewidth is small
  - uses built-in optimization to get the best obtained abstraction after 35 seconds

Implementation nestHDB

Framework based on dpdb using database systems [FHecherThierWoltran'21]

- Why databases?
  1. Efficient table manipulation operations (selection, projection and joins)
  2. Convenient representation of table algorithms
- written in Python3, multi-core support, uses PostgreSQL 12 on ramdisk
- open-source: https://github.com/hmarkus/dp_on_dbs/tree/nesthdb
- relies on SAT solver picosat, MC solver MC and PM solver projMC for search-intense subproblems $\rightsquigarrow$ Hybrid Solving
- "Good" nested primal graphs (abstractions) are approximated via a logic program
  - prefers abstractions that are large, but not dense, i.e., whose treewidth is small
  - uses built-in optimization to get the best obtained abstraction after 35 seconds

Implementation nestHDB

Framework based on dpdb using database systems [FHecherThierWoltran'21]

- Why databases?
  1. Efficient table manipulation operations (selection, projection and joins)
  2. Convenient representation of table algorithms
- written in Python3, multi-core support, uses PostgreSQL 12 on ramdisk
- open-source: https://github.com/hmarkus/dp_on_dbs/tree/nesthdb
- relies on SAT solver picosat, MC solver MC and PM solver projMC for search-intense subproblems $\rightsquigarrow$ Hybrid Solving
- "Good" nested primal graphs (abstractions) are approximated via a logic program
  - prefers abstractions that are large, but not dense, i.e., whose treewidth is small
  - uses built-in optimization to get the best obtained abstraction after 35 seconds

Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_\downarrow$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then return** $\begin{pmatrix} cnt \\ \hline 1 \end{pmatrix}$
2 **else if** intr and $a \in V_t$ new **then**
3 **return** $\sigma_{row \models F_t}\left(T_\downarrow \bowtie \begin{pmatrix} a \\ \hline 0 \\ 1 \end{pmatrix}\right)$
4 **else if** rem, and $a \notin V_t$ removed **then**
5 **return**

x_1	x_2	~~a~~	~~cnt~~	
0	0	~~0~~	~~c^0_{00}~~	$\Rightarrow \sum_{c^0_{00}+c^1_{00}}$
~~0~~	~~0~~	~~1~~	~~c^1_{00}~~	
0	1	~~0~~	~~c^0_{01}~~	$\Rightarrow \sum_{c^0_{01}+c^1_{01}}$
~~0~~	~~1~~	~~1~~	~~c^1_{01}~~	
...				

6 **else if** join **then**
7 **return** $T_{\downarrow l} \bowtie T_{\downarrow r}$ + "fix cnt (multiply

Nested Dynamic Programming for PMC

1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$

b

Nested Dynamic Programming for PMC

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

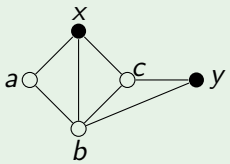
1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$\{x,\ y\}$

$\{b\}$

Nested Dynamic Programming for PMC

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



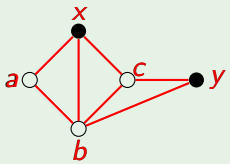
1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

b

$x,\ y$

Nested Dynamic Programming for PMC

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



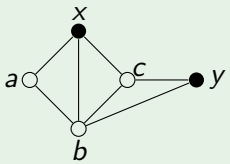
1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$

x	y	#
0	0	1
0	1	1
1	0	1
1	1	0

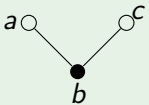
Nested Dynamic Programming for PMC

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



1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$

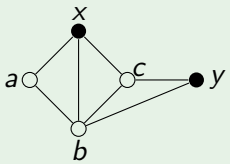


b

x	y	#
0	0	1
0	1	1
1	0	1
1	1	0

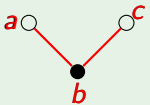
Nested Dynamic Programming for PMC

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



1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$



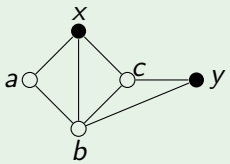
b

x	y	#
0	0	1
0	1	1
1	0	1
1	1	0

b	#
1	1

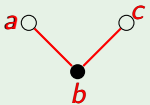
Nested Dynamic Programming for PMC

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



1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$



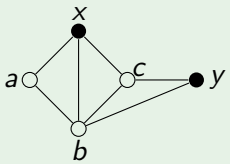
b

x	y	#
0	0	1
0	1	1
1	0	1
1	1	0

b	#
1	1

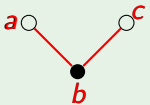
Nested Dynamic Programming for PMC

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



1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$



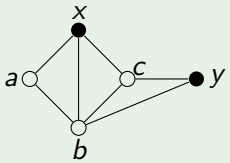
b

x	y	#
0	0	1
0	1	1
1	0	1
1	1	0

b	#
1	1

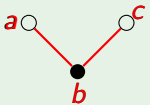
Nested Dynamic Programming for PMC

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



1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$



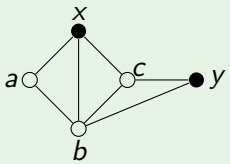
b

x	y	#
0	0	1
0	1	1
1	0	1
1	1	0

b	#

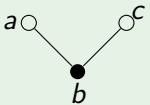
Nested Dynamic Programming for PMC

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



1. Decompose nested primal graph $N_F^{\{x,y\}}$
2. Solve subproblems
3. Decompose & Solve using $N_{F[I]}^{\{b\}}$ for assignment I to $\{x, y\}$
4. Combine solutions

$x,\ y$



b

x	y	#
0	0	1
0	1	1
1	0	1
1	1	0

b	#
1	1

Algorithm for Nested Dynamic Programming

Dynamic Programming for SAT SamerSzeider 10

1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions

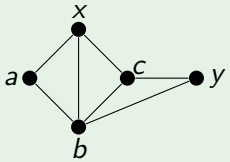
Dynamic Programming for SAT SamerSzeider 10

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

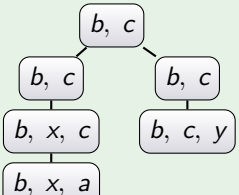
1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions

Dynamic Programming for SAT SamerSzeider 10

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

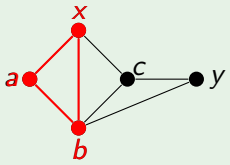


1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions

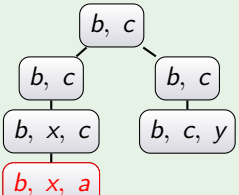


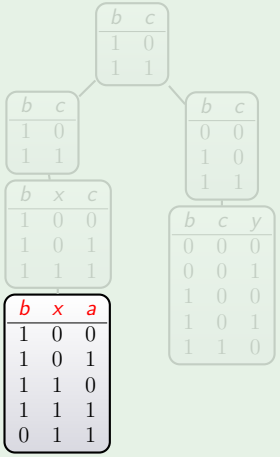
Dynamic Programming for SAT SamerSzeider 10

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



1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions

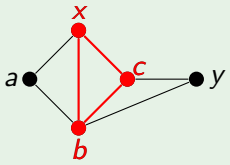




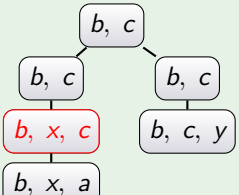
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

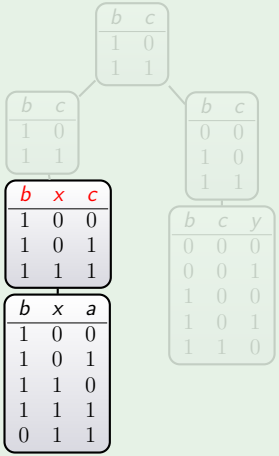
Dynamic Programming for SAT SamerSzeider 10

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



1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions





b	c
1	0
1	1

b	c
1	0
1	1

b	c
0	0
1	0
1	1

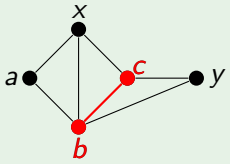
b	x	c
1	0	0
1	0	1
1	1	1

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

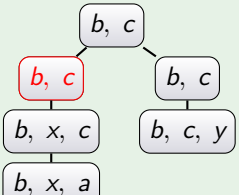
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

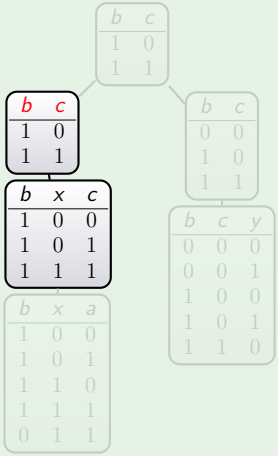
Dynamic Programming for SAT SamerSzeider 10

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



1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions





b	c
1	0
1	1

b	c
0	0
1	0
1	1

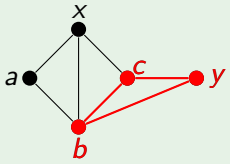
b	x	c
1	0	0
1	0	1
1	1	1

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

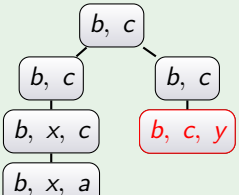
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

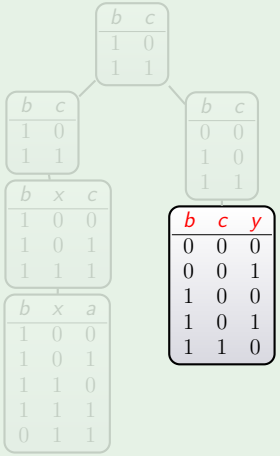
Dynamic Programming for SAT SamerSzeider 10

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



1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions





b	c
1	0
1	1

b	c
1	0
1	1

b	c
0	0
1	0
1	1

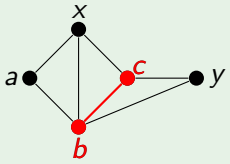
b	x	c
1	0	0
1	0	1
1	1	1

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

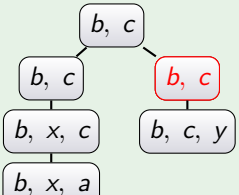
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

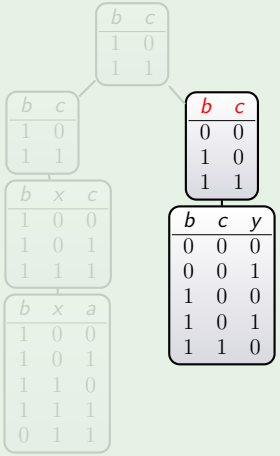
Dynamic Programming for SAT SamerSzeider 10

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



1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions





b	c
1	0
1	1

b	c
1	0
1	1

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

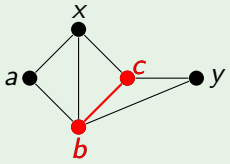
b	x	a
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

b	c
0	0
1	0
1	1

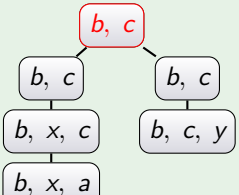
b	c	y
0	0	0
0	0	1
1	0	0
1	0	1
1	1	0

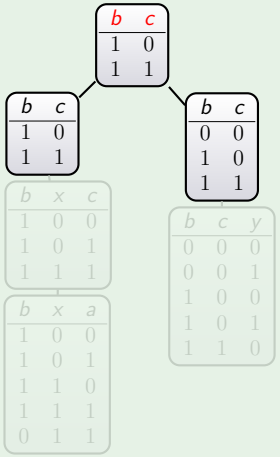
Dynamic Programming for SAT SamerSzeider 10

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



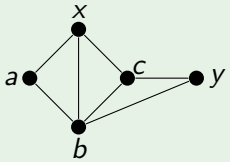
1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions



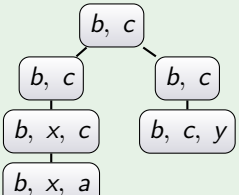


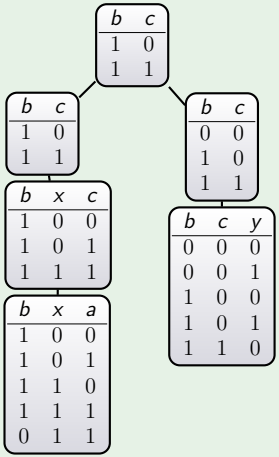
Dynamic Programming for SAT SamerSzeider 10

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



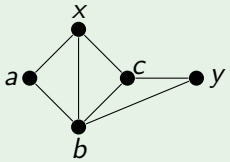
1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions



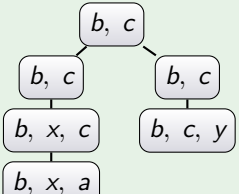


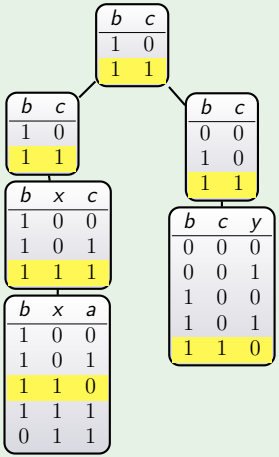
Dynamic Programming for SAT SamerSzeider 10

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



1. Decompose primal graph P_F
2. Solve subproblems
3. Combine solutions

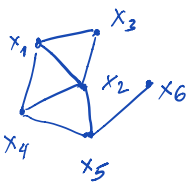




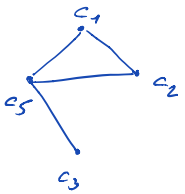
Graphical Models

$F = \{c_1, c_2, c_3, c_4, c_5\}$ $c_1 = \{x_1, x_2, x_3\}$ $c_2 = \{\bar{x}_1, x_4\}$ $c_3 = \{x_2, x_5\}$
$c_4 = \{\bar{x}_5, x_6\}$ $c_5 = \{x_1, x_2, \bar{x}_4\}$

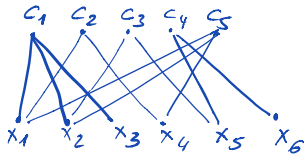
Primal Graph



Dual Graph



Incidence Graph



$inc\text{-}tw(F) \leq primal\text{-}tw(F) + 1$

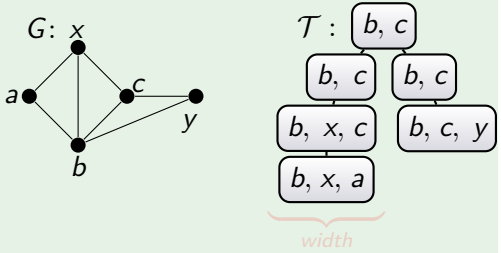
$inc\text{-}tw(F) \leq dual\text{-}tw(F) + 1$

primal tw and inc-tw - incomp.
(large clause $\Rightarrow$ high prim tw
variable that occ. $\Rightarrow$ high dual tw
in many cls

Width/Decomposition Measures

- treewidth and branchwidth Robertson&Seymour 1986
  (size of largest vertex set in a tree decomposition of the graph)
- clique-width Courcelle, Engelfriet, Rozenberg 1990,
  (can be bounded for dense graphs)
  can also be defined for directed graphs
- hypertree width Gottlob, Leone, Scarcello 2002
  (width function instead of size)

Tree Decomposition $\mathcal{T}$ of G



(#)SAT

width	Solving*	Reference
htw	paraNP-c	Samer, Szeider 2010
undir incidence cw	XP	Ordyniak, Paulusma, Szeider 2010; Slivovsky, Szeider 2013
dir incidence cw	FPT	Courcelle, Makowsky, Rotics 2001
dual tw	FPT $\mathcal{O}^*(2^k)$	Samer, Szeider 2010
incidence tw	FPT $\mathcal{O}^*(4^k)$ $\mathcal{O}^*(2^k)$	Fischer, Makowsky, Ravve 2008; Samer, Szeider 2010 Slivovsky, Szeider 2020
primal tw	FPT $\mathcal{O}^*(2^k)$	Freuder 1985; Alekhnovich, Razborov 2002 Bacchus, Dalamao, Pitassi 2003

*: Abuse of notation for counting

(Fractional) Hypertree Decompositions

Definition ((Fractional) Edge Cover)

$H = (V, E)$ Hypergraph and $S \subseteq V$ set

- A (fractional) edge cover of S is mapping $\gamma : E \to [0, 1]$ s.t. for each $v \in S$: $\sum_{e \in E,\, v \in e} \gamma(e) \geq 1$.
- Weight of γ: $\sum_{e \in E} \gamma(e)$
- $_H(S)$, is the min. weight over all its (fractional) edge covers

Definition ((Fractional) hypertree decomposition)

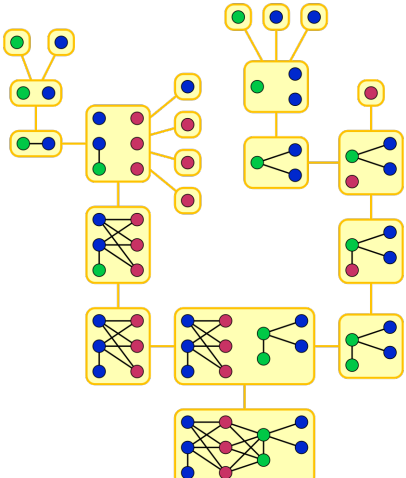
Triple $\mathcal{F} = (T, \chi, \gamma)$ where (T, χ) is a tree decomposition of H and γ is a mapping that assigns each $t \in V(T)$ a (fractional) edge cover $\lambda(t)$ of $\chi(t)$
The width of $\mathcal{F}$ is the largest weight of the (fractional) edge covers $\lambda(t)$ over all $t \in V(T)$. The (fractional) hypertree width (H) of H is the smallest width over all (fractional) hypertree decompositions of H.

Clique-Width

Closely related to treewidth (can be bounded for dense graphs)

clique-width: minimum number of labels needed to construct G by:

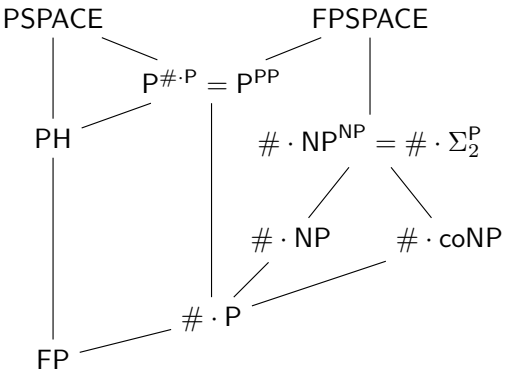
1. Creation of a new vertex v with label i (noted $i(v)$)
2. Disjoint union of two labeled graphs G and H (denoted $G \oplus H$)
3. Joining by an edge every vertex labeled i to every vertex labeled j where $i \neq j$ (denoted $\nu(i,j)$)
4. Renaming label i to label j (denoted $\rho(i,j)$)



Outline

3) Computational Complexity

Computational Complexity

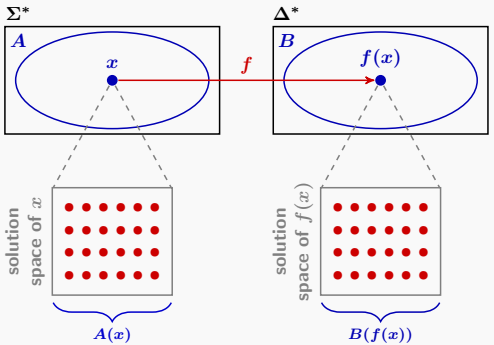


- $\bigcup_{k \in \mathbb{N}} \Delta_k^P = \mathrm{PH}$
- $\mathrm{NP} \subseteq \Delta_2^P = P^{\mathrm{NP}}$
- Approximate Counting: approx-$\# \cdot P \subseteq \mathrm{BPP}^{\mathrm{NP}} \subseteq \Sigma_3^P$

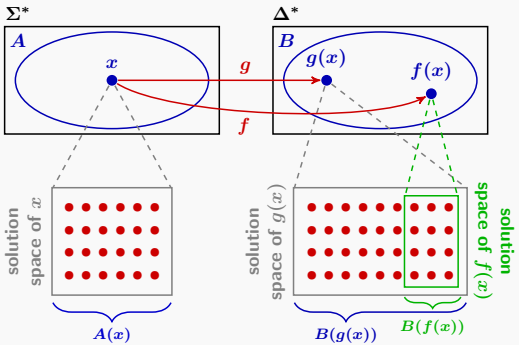
- $\# \cdot$ coNP: complete wrt. subtractive reductions
- others/w: complete wrt. parsimonious reductions
- Framework of Hemaspaandra/Vollmer 1995 and Durand/Hermann/Kolaitis 2005.

Computational Complexity

Reductions



parsimonious reductions maintain cardinalities of the solution spaces, so $|A(x)| = |B(f(x))|$



subtractive reductions recalculate cardinalities: $|A(x)| = |B(g(x))| - |B(f(x))|$

Quantified Boolean Formulas (QBFs)

Definition (QBFSAT$_\ell$)

An instance of QBFSAT$_\ell$ is of the form $Q = Q_1 X_1 Q_2 X_2 \ldots Q_\ell X_\ell . F$ where:

- Matrix F is a Boolean formula in CNF (if $Q_\ell = \exists$), or in DNF (if $Q_\ell = \forall$)
- Quantifier $Q_i \in \{\exists, \forall\}$ such that $Q_i \neq Q_{i+1}$, and $X_i \neq X_{i+1}$
- F is closed: every variable in F occurs in exactly one set X_i.

Question: Is Q valid?

QBF semantics are recursively defined:

- $\forall X.Q$ is valid if $Q[\iota]$ is valid for every truth assignment $\iota : X \to \{1, 0\}$
- $\exists X.Q$ is valid if $Q[\iota]$ is valid for some truth assignment $\iota : X \to \{1, 0\}$

Example:

$$Q = \exists w, x. \forall y, z. \underbrace{(w \wedge x \wedge \neg y)}_{d_1} \vee \underbrace{(\neg w \wedge \neg x \wedge y)}_{d_2} \vee \underbrace{(w \wedge y \wedge \neg z)}_{d_3} \vee \underbrace{(w \wedge y \wedge z)}_{d_4}$$

Quantified Boolean Formulas (QBFs)

Definition (QBFSAT$_\ell$)

An instance of QBFSAT$_\ell$ is of the form $Q = Q_1 X_1 Q_2 X_2 \ldots Q_\ell X_\ell . F$ where:

- Matrix F is a Boolean formula in CNF (if $Q_\ell = \exists$), or in DNF (if $Q_\ell = \forall$)
- Quantifier $Q_i \in \{\exists, \forall\}$ such that $Q_i \neq Q_{i+1}$, and $X_i \neq X_{i+1}$
- F is closed: every variable in F occurs in exactly one set X_i.

Question: Is Q valid?

QBF semantics are recursively defined:

- $\forall X.Q$ is valid if $Q[\iota]$ is valid for every truth assignment $\iota : X \to \{1, 0\}$
- $\exists X.Q$ is valid if $Q[\iota]$ is valid for some truth assignment $\iota : X \to \{1, 0\}$

Example:

$$Q = \exists w, x. \forall y, z. \underbrace{(w \wedge x \wedge \neg y)}_{d_1} \vee \underbrace{(\neg w \wedge \neg x \wedge y)}_{d_2} \vee \underbrace{(w \wedge y \wedge \neg z)}_{d_3} \vee \underbrace{(w \wedge y \wedge z)}_{d_4}$$

Quantified Boolean Formulas (QBFs)

Definition (QBFSAT$_\ell$)

An instance of QBFSAT$_\ell$ is of the form $Q = Q_1 X_1 Q_2 X_2 \ldots Q_\ell X_\ell . F$ where:

- Matrix F is a Boolean formula in CNF (if $Q_\ell = \exists$), or in DNF (if $Q_\ell = \forall$)
- Quantifier $Q_i \in \{\exists, \forall\}$ such that $Q_i \neq Q_{i+1}$, and $X_i \neq X_{i+1}$
- F is closed: every variable in F occurs in exactly one set X_i.

Question: Is Q valid?

QBF semantics are recursively defined:

- $\forall X.Q$ is valid if $Q[\iota]$ is valid for every truth assignment $\iota : X \to \{1, 0\}$
- $\exists X.Q$ is valid if $Q[\iota]$ is valid for some truth assignment $\iota : X \to \{1, 0\}$

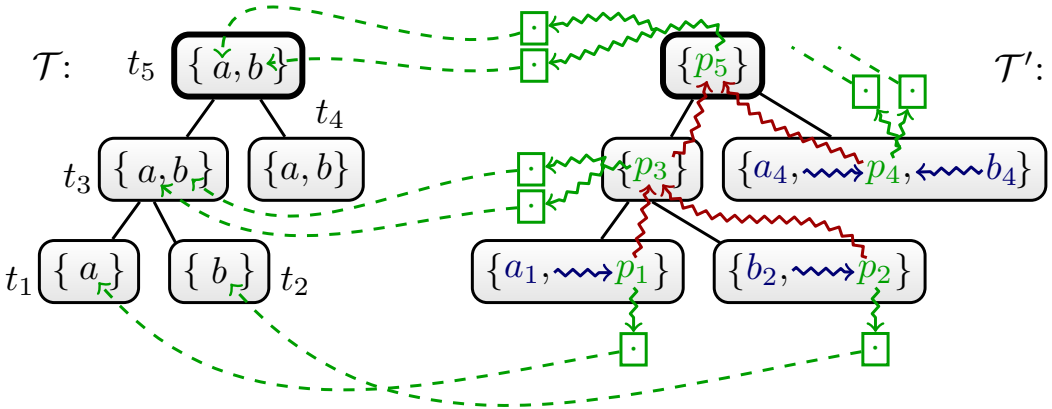
Example:

$$Q = \exists w, x. \forall y, z. \underbrace{(w \wedge x \wedge \neg y)}_{d_1} \vee \underbrace{(\neg w \wedge \neg x \wedge y)}_{d_2} \vee \underbrace{(w \wedge y \wedge \neg z)}_{d_3} \vee \underbrace{(w \wedge y \wedge z)}_{d_4}$$

Methodology on Lower Bounds

Standard approach	Our approach
(SAT, k)	$(\mathsf{QBFSAT}_\ell, k)$
$\downarrow_?$	$\downarrow_{\mathcal{R}}$
$(\mathcal{P}, \log^\ell(n))$	$(\mathcal{P}, k)$

Outline of DG Reduction $\mathcal{R}$ for QBFSAT



$\mathcal{R}$ consists of **guess**, **propagate**, and **check** parts, thereby heavily relying on **pointers**

Reduction $\mathcal{R}$ by Example

$Q = \exists w, x. \forall y, z. (w \wedge x \wedge \neg y) \vee (\neg w \wedge \neg x \wedge y) \vee (w \wedge y \wedge \neg z) \vee (w \wedge y \wedge z)$

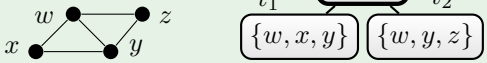


Figure: Primal graph P_Q of Q and a TD $\mathcal{T}$ of P_Q.

Reduction $\mathcal{R}$ by Example

$Q = \exists w, x. \forall y, z. (w \wedge x \wedge \neg y) \vee (\neg w \wedge \neg x \wedge y) \vee (w \wedge y \wedge \neg z) \vee (w \wedge y \wedge z)$

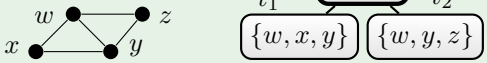


Figure: Primal graph P_Q of Q and a TD $\mathcal{T}$ of P_Q.

Reduction $\mathcal{R}$ by Example

$Q = \exists w, x. \forall y, z. (w \wedge x \wedge \neg y) \vee (\neg w \wedge \neg x \wedge y) \vee (w \wedge y \wedge \neg z) \vee (w \wedge y \wedge z)$

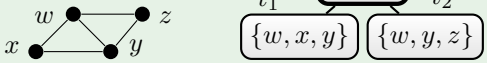
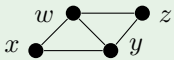


Figure: Primal graph P_Q of Q and a TD $\mathcal{T}$ of P_Q.

$$\mathcal{R}(Q, \mathcal{T}) = \exists w_{t_1}, w_{t_2}, x_{t_1}. \forall y_{t_1}, z_{t_2}, \underbrace{b^0_{t_1}, b^1_{t_1}, b^0_{t_2}, b^1_{t_2}, b^0_{t_3}}_{\mathcal{T}}. \exists y_{t_2}, \underbrace{v_{t_1}, \ldots, v_{t_3}}_{\mathcal{T}} \cdots$$

Reduction $\mathcal{R}$ by Example

$Q = \exists w, x. \forall y, z. (w \wedge x \wedge \neg y) \vee (\neg w \wedge \neg x \wedge y) \vee (w \wedge y \wedge \neg z) \vee (w \wedge y \wedge z)$



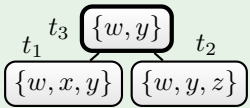


Figure: Primal graph P_Q of Q and a TD $\mathcal{T}$ of P_Q.

Guess part for w:

$$w_{t_1} \wedge \underbrace{\neg b^0_{t_1} \wedge \neg b^1_{t_1}}_{wt_1} \longrightarrow v_{t_1}, \quad \neg w_{t_1} \wedge \underbrace{\neg b^0_{t_1} \wedge \neg b^1_{t_1}}_{wt_1} \longrightarrow \neg v_{t_1},$$

$$w_{t_2} \wedge \underbrace{\neg b^0_{t_2} \wedge \neg b^1_{t_2}}_{wt_2} \longrightarrow v_{t_2}, \quad \neg w_{t_2} \wedge \underbrace{\neg b^0_{t_2} \wedge \neg b^1_{t_2}}_{wt_2} \longrightarrow \neg v_{t_2}$$

Reduction $\mathcal{R}$ by Example

$Q = \exists w, x. \forall y, z. (w \wedge x \wedge \neg y) \vee (\neg w \wedge \neg x \wedge y) \vee (w \wedge y \wedge \neg z) \vee (w \wedge y \wedge z)$

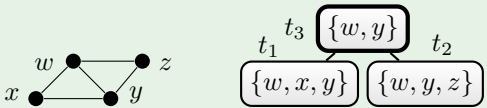


Figure: Primal graph P_Q of Q and a TD $\mathcal{T}$ of P_Q.

Propagate part for w and t_3:

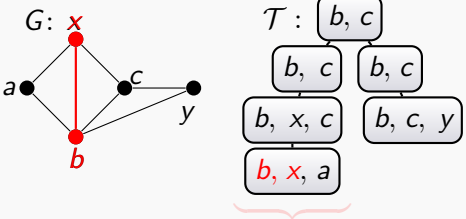
$$v_{t_3} \wedge \underbrace{\neg b^0_{t_3}}_{wt_3} \wedge \underbrace{\neg b^0_{t_1} \wedge \neg b^1_{t_1}}_{wt_1} \longrightarrow v_{t_1}, \quad \neg v_{t_3} \wedge \underbrace{\neg b^0_{t_3}}_{wt_3} \wedge \underbrace{\neg b^0_{t_1} \wedge \neg b^1_{t_1}}_{wt_1} \longrightarrow \neg v_{t_1}$$

$$v_{t_3} \wedge \underbrace{\neg b^0_{t_3}}_{wt_3} \wedge \underbrace{\neg b^0_{t_2} \wedge \neg b^1_{t_2}}_{wt_2} \longrightarrow v_{t_2}, \quad \neg v_{t_3} \wedge \underbrace{\neg b^0_{t_3}}_{wt_3} \wedge \underbrace{\neg b^0_{t_2} \wedge \neg b^1_{t_2}}_{wt_2} \longrightarrow \neg v_{t_2}$$

Structural Decompositions

Tree Decompositions and Treewidth

- Most prominent graph invariant
- Small treewidth indicates tree-likeness and sparsity



Dynamic Programming: Parallel Execution

How to parallelize DP?

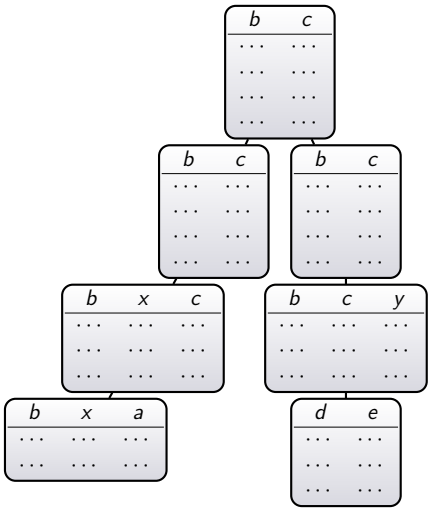
1. Compute tables for multiple nodes in parallel

⇒ 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 FHecherRolandZisserESA'18,CP'19,'21

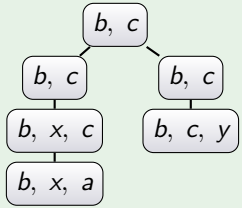
Why: computation of rows are independent



Relational Algebra FHecherWoltranThier TPLP'22

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows



"Local formula" F_t clauses whose variables are contained in the bag

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** **return** $\begin{pmatrix} cnt \\ 1 \end{pmatrix}$
2 **else if** intr and $a \in V_t$ new **then**
3 **return** $\sigma_{row \models F_t}\left(T_{\downarrow} \times \begin{pmatrix} a \\ 0 \\ 1 \end{pmatrix}\right)$
4 **else if** rem, and $a \notin V_t$ removed **then**
5 **return**

$$\left(\begin{array}{cccc|c} x_1 & x_2 & \cancel{a} & cnt & \\ \hline 0 & 0 & 0 & \cancel{c^0_{00}} & \Rightarrow \sum_{c^0_{00}+c^1_{00}} \\ \cancel{0} & \cancel{0} & \cancel{1} & \cancel{c^1_{00}} & \\ 0 & 1 & 0 & \cancel{c^1_2} & \Rightarrow \sum_{c^0_{01}+c^1_{01}} \\ \cancel{0} & \cancel{1} & \cancel{1} & \cancel{c^1_2} & \\ \dots & & & & \end{array}\right)$$

6 **else if** join **then**
7 **return** $T_{\downarrow l} \bowtie T_{\downarrow r}$ + "fix cnt (multiply

Paralleling Dynamic Programming

How to parallelize DP?

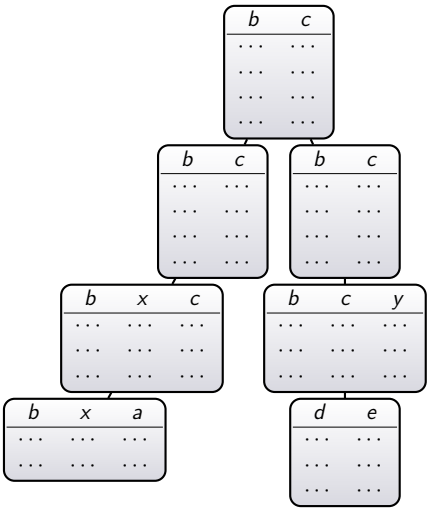
1. Compute tables for multiple nodes in parallel

⇒ 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



Paralleling Dynamic Programming

How to parallelize DP?

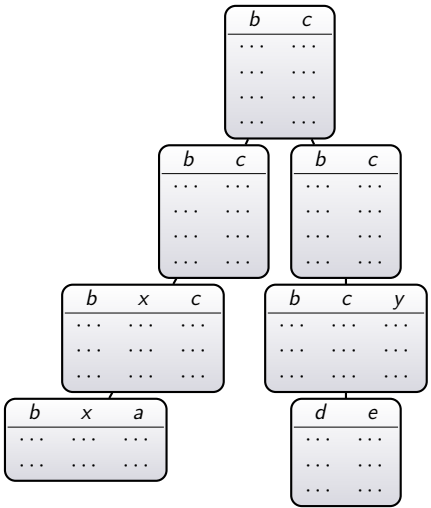
1. Compute tables for multiple nodes in parallel

⇒ 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

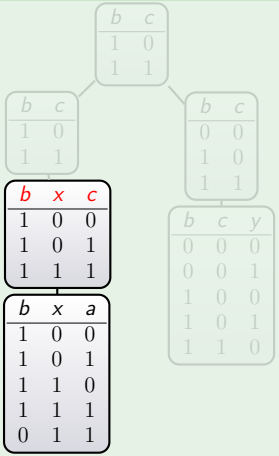


Formulate it in SQL FHecherWoltranThier TPLP'22

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_\downarrow$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`



Runtime: Summary

		solver	0-20	21-30	31-40	41-50	51-60	>60	best	unique	$\sum$	time[h]
preprocessed by pmc	1.	miniC2D	1193	29	**10**	2	1	7	11	0	**1242**	**68.77**
	2.	dpdb	1193	31	7	2	0	0	2	1	1233	70.44
	3.	gpusat2	**1196**	**32**	1	0	0	0	154	1	1229	71.27
	4.	d4	1163	20	**10**	2	**4**	28	33	1	1227	76.86
	5.	countAntom	1141	18	**10**	5	**4**	13	102	0	1191	84.39
	6.	dpdb pg9	1159	19	5	2	0	0	0	0	1184	100.99
	7.	c2d	1124	31	**10**	3	3	10	24	0	1181	84.41
	8.	ganak	1031	16	10	2	**4**	29	**633**	0	1092	107.25
	9.	sharpSAT	1029	16	**10**	2	**4**	30	80	0	1091	106.88
	10.	sdd	1014	4	7	1	0	2	0	0	1028	124.23
	11.	sts	927	4	8	**7**	5	**52**	35	**21**	1003	128.43
	12.	dsharp	853	3	7	2	0	0	29	0	865	157.87
	13.	cnf2eadt	799	3	7	2	0	7	157	0	818	170.17
	14.	approxmc 3	794	3	7	2	0	6	1	0	812	173.35
	15.	bdd_minisat	791	4	1	0	0	0	46	0	796	175.09

Outline

2) Application to Bayesian Reasoning

Uncertainty in AI

Probability Distribution

= Qualitative + Quantitative

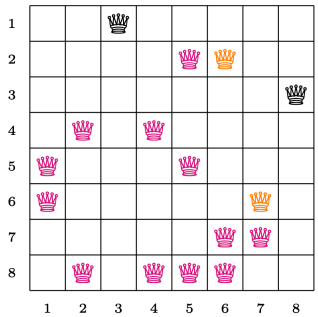
Probability Distribution

= Logic + Counting

Application: Comprehending Search Spaces

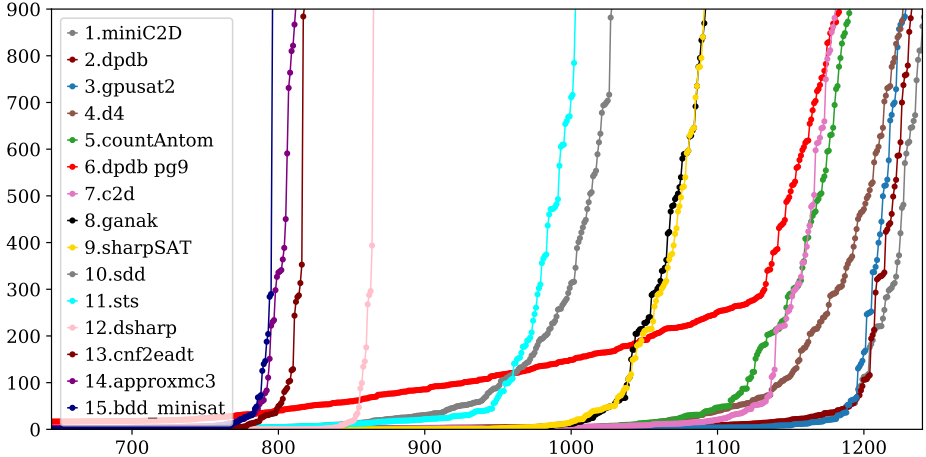
	2		5		1		9	
8			2		3			6
	3			6			7	
6	*6*	1		*2*	*2|6*			
5	4			*2*		*8*	1	9
				2|5	*2|5*	7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Is the solution unique? How many solutions are allowed in one field?



Which moves (queens) have the least $(1/4)$ / most $(3/4)$ impact?

Runtime: Asymptotical Consideration

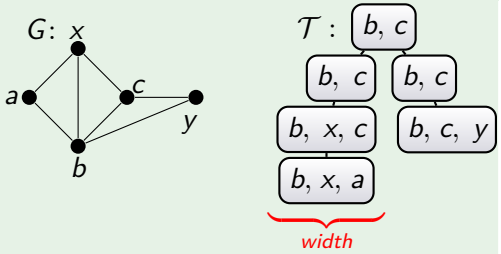


Outline

1. Treewidth and Tree Decompositions

Tree Decompositions

Tree Decomposition $\mathcal{T}$ of G



Definition

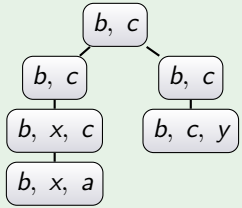
A tree decomposition is a tree obtained from an input graph s.t.

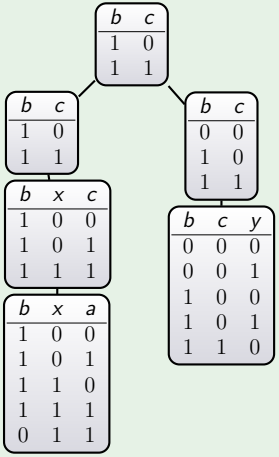
1. Each vertex must occur in some bag
2. For each edge, there is a bag containing both endpoints
3. Connected: Tree "restricted" to any vertex must be connected

Solving #SAT [SamerSzeider10]

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows





"Local formula" F_t clauses whose variables are contained in the bag

Solving #SAT

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

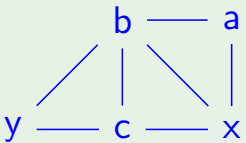
$$\mathsf{Mod}(F) = \{ \{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \{b, c, x\}, \{a, b, c, x\}, \{b, y\}, \{a, b, y\} \}$$

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

Solving #SAT

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

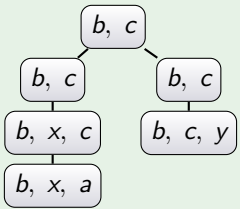
1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows



Employing Decompositions for Model Counting

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

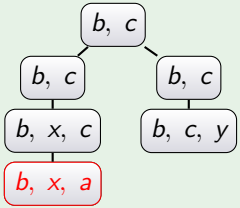
1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

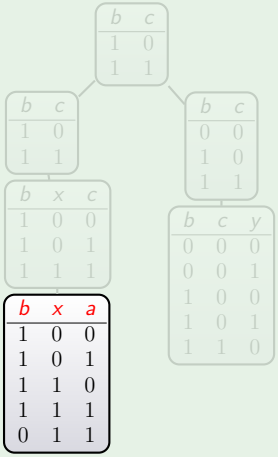


Employing Decompositions for Model Counting

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows



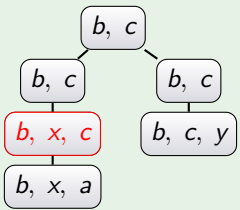


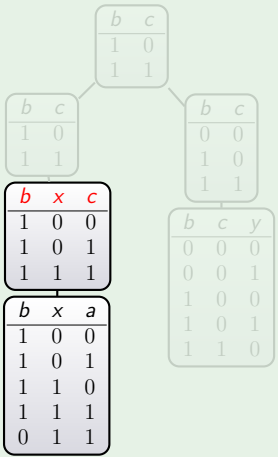
"Local formula" F_t clauses whose variables are contained in the bag

Employing Decompositions for Model Counting

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

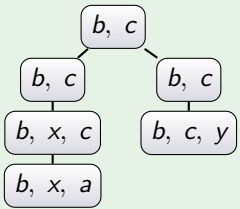


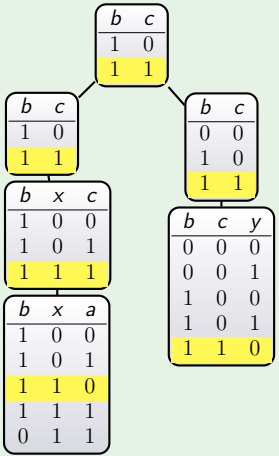


Employing Decompositions for Model Counting

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

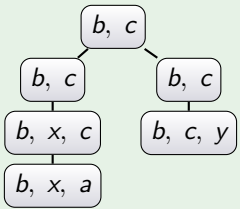


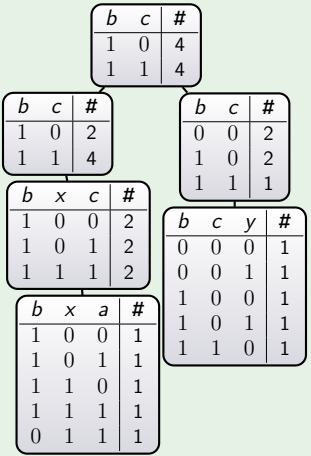


Employing Decompositions for Model Counting

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows





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

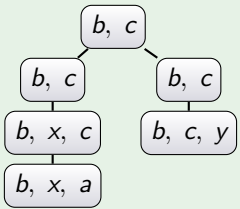
b	c	y	#
0	0	0	1
0	0	1	1
1	0	0	1
1	0	1	1
1	1	0	1

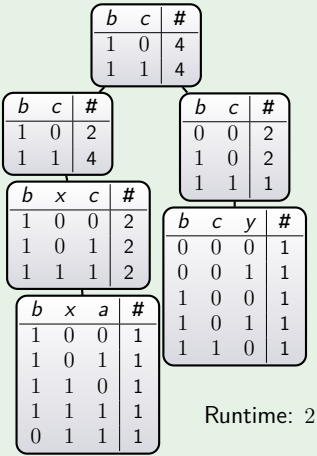
b	x	a	#
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1
0	1	1	1

Employing Decompositions for Model Counting

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows





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

Runtime: $2^{O(\text{treewidth})} \cdot \text{poly}(|F|)$

Massively Parallel Execution

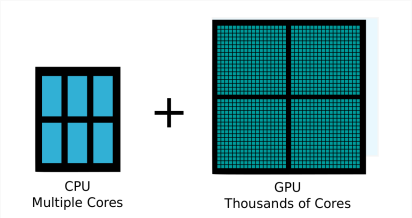
Implementation FHecherRolandZisser ESA'18,CP'19,CP'21

- You need to get your hands dirty
- Architecture matters (first OpenCL / latest version CUDA)
- Implementation for inc-tw rarely pays off (inctw+1 <= primtw; can be exponentially smaller)

"Constants Matter"

- Boils sometimes down to bit fiddling
- Use low level data structures
- Do as much as possible on the GPU
- Efficient heuristics are key

$\Rightarrow$ Works well



Empirical Work

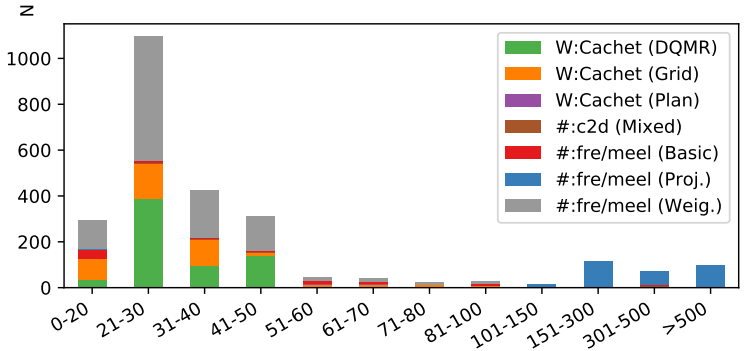
Starting Point

- Research a set of instances and analyze them
- Standard instance set

Disclaimer (following slides)

- Older instance set (2019)
- Slightly different picture on other instances
- 2021 solvers improved
- (lack of resources to rerun frequently)

Empirical Work: Distribution of Upper Bounds on Primal Tw



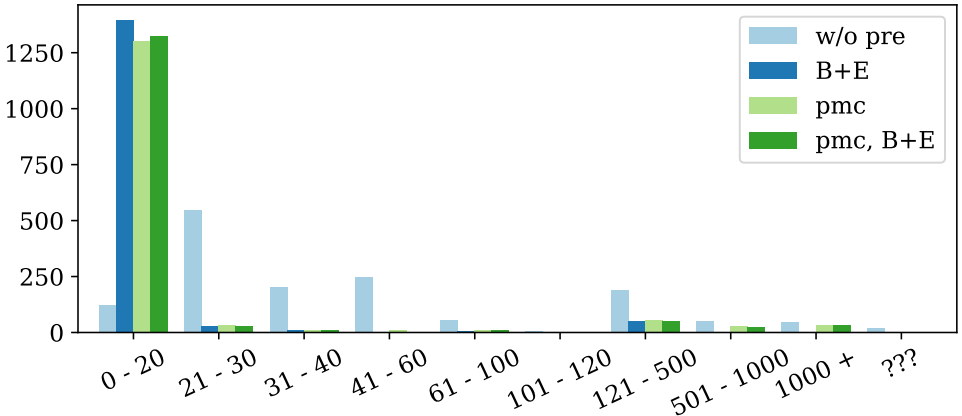
Decomposition Heuristic:

- 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...

#SAT: Width Comparison (w/o Preprocessing)



$\Rightarrow$ Produce decompositions of significantly smaller width

Outline

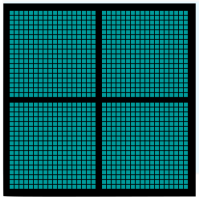
4. Dynamic Programming (GPU)

A GPU-based #SAT-solver

OR how to go massively parallel?

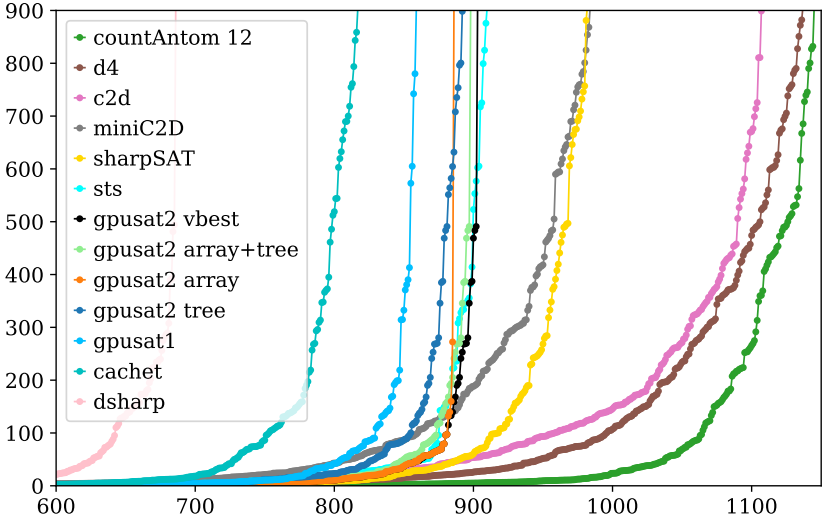
CPU
Multiple Cores

+

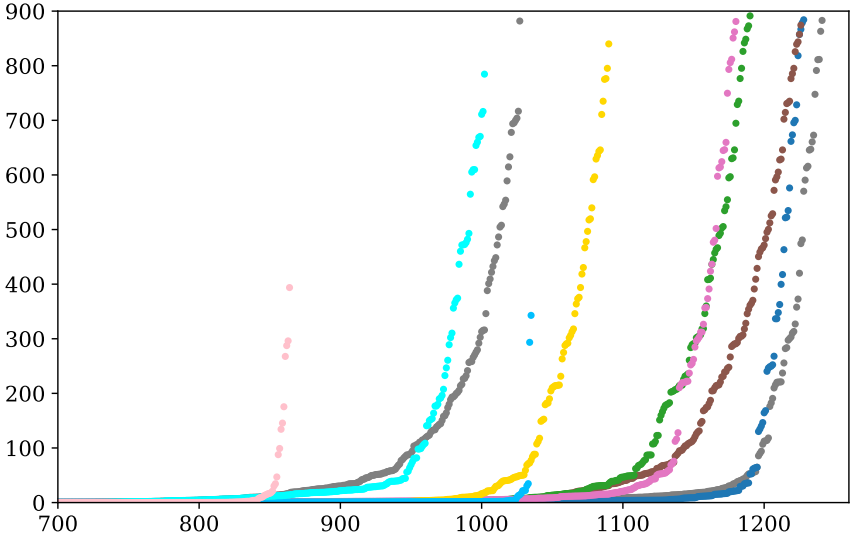


GPU
Thousands of Cores

#SAT: Runtime Results (wo. Preprocessing)



#SAT: Runtime Results (w. Preprocessing)



→ Techniques pay off after preprocessing

Lessons Learned?

Implementation

- You need to get your hands dirty
- Architecture matters (first OpenCL / latest version CUDA)
- Implementation for inc-tw rarely pays off
  (inctw+1 <= primtw; can be exponentially smaller)

"Constants Matter"

- Boils sometimes down to bit fiddling
- Use low level data structures
- Avoid copying (VRAM is very slow)
- Efficient heuristics are key

$\Rightarrow$ Works surprisingly well

Outline

5. Dynamic Programming (Databases)

Paralleling Dynamic Programming

How to parallelize DP?

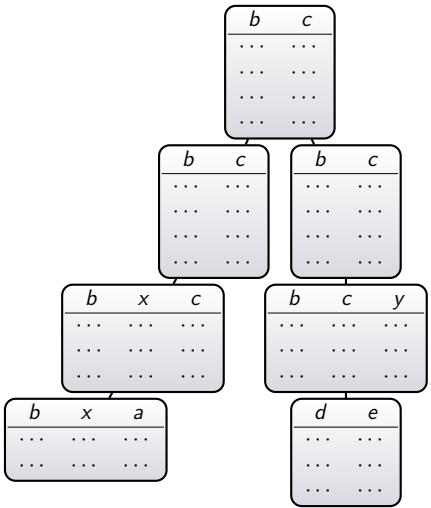
1. Compute tables for multiple nodes in parallel

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

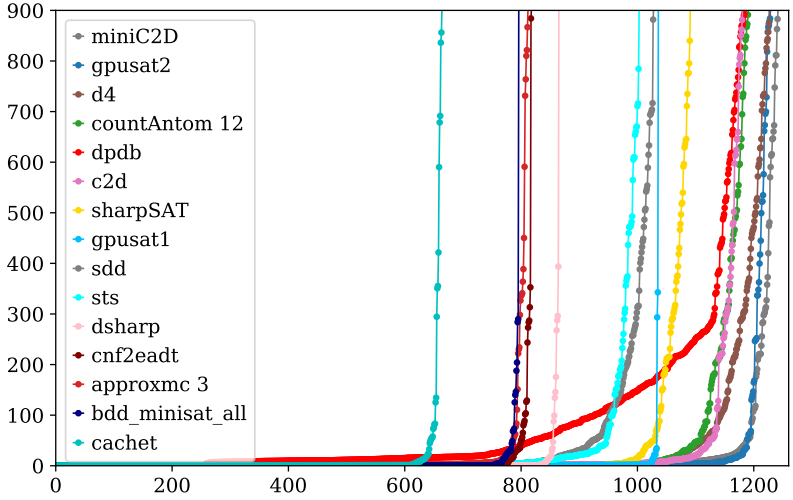
2. Distribute computation of rows among different computation units

⇒ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



#SAT (Runtime)


miniC2D
gpusat2
d4
countAntom 12
dpdb
c2d
sharpSAT
gpusat1
sdd
sts
dsharp
cnf2eadt
approxmc 3
bdd_minisat_all
cachet
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200

Lessons Learned?

Implementation

- Works surprisingly well (gives a framework for many problems)
- Has the obvious theoretical limitations
- Again: details matter
- Difference between storing data in memory and on the disk large (just a constant)
- Don't try to compete with years of engineering without need (algorithms inside databases)
- Treewidth might not be enough

Solving Problem on GPU or Employing Databases

Implementation

- You need to get your hands dirty
- Efficient heuristics are key
- Don't try to compete with years of engineering without need (algorithms inside databases)
- Constants Matter
- Works surprisingly well (gives a framework for many problems)

$\Rightarrow$ Parameterized Complexity works surprisingly well

Personal Observations

When is a solver seen as good?

- "Beats" all state-of-the-art existing solvers?
- Thorough evaluation of theoretical properties to practice not interesting?

Challenge

- Modern solvers highly engineered (not just data structures, but understanding of the underlying architecture; memory cachelines etc.; rarely documented in papers)
- Understanding practical solving, requires more fine-grained measures
- Solver engineer vs. programming wizard
- Be more forthcoming with accepting papers on implementations

  Why: Science lives from new knowledge.

  Standards required (studies in medical sciences, experimental physics...)

Summary

Applications in model-based AI

- Abstract Argumentation (FHecherMeier) [AAAI'19]
- Description Logics (FHecherMeierMahmood) [AAAI'21]
- QCSP (FHecherKieler) [CP'20]
- Conditional Lower Bounds for QBF wpb tw
  Height of the exponentials depends on no qf alternations (FHecherPfandler) [LICS'20]

Experimental Evaluation of nestHDB

We ran two sets of benchmarks

- MC on about 1,500 publicly available instances
- PM on about 600 publicly available instances

Thereby we compared both publicly available single-core as well as multi-core solvers

- Measure: Wallclock time
- Timeout: 900s, Memout: 16GB, CPU Core Limit: 12 Cores
- Hardware: Intel Xeon E5-2650, 2.2GHz, 256GB RAM, Linux 4.4.0-139

Experimental Evaluation of nestHDB

We ran two sets of benchmarks

- MC on about 1,500 publicly available instances
- PM on about 600 publicly available instances

Thereby we compared both publicly available single-core as well as multi-core solvers

- Measure: Wallclock time
- Timeout: 900s, Memout: 16GB, CPU Core Limit: 12 Cores
- Hardware: Intel Xeon E5-2650, 2.2GHz, 256GB RAM, Linux 4.4.0-139

Lessons Learned?

Implementation

- Works surprisingly well (gives a framework for many problems)
- Has the obvious theoretical limitations
- Again: details matter
- Difference between storing data in memory and on the disk large (just a constant)
- Don't try to compete with years of engineering without need (algorithms inside databases)
- Treewidth might not be enough

Solving #SAT [SamerSzeider10]

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

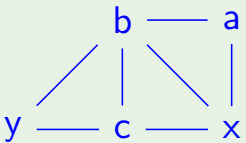
$$\mathsf{Mod}(F) = \{ \{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \{b, c, x\}, \{a, b, c, x\}, \{b, y\}, \{a, b, y\} \}$$

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

Solving #SAT [SamerSzeider10]

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows



Algorithm for Primal Graph

In: Node t, bag χ_t, clauses F_t, sequence C of tables.
Out: Table tab_t

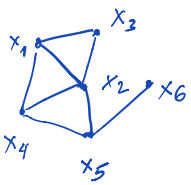
if $\mathrm{type}(t) = \mathrm{leaf}$ **then**
  $\mathrm{tab}_t \leftarrow \{\emptyset\}$
else if $\mathrm{type}(t) = \mathrm{intr}$ *and* $a \in \chi_t \setminus \chi_{t'}$, **then**
  $\mathrm{tab}_t \leftarrow \left\{\tau \cup \{a\} \mid \tau \in \mathrm{tab}'', \tau \cup \{a\} \models F_t\right\} \cup$
  $\left\{\tau \mid \tau \in \mathrm{tab}'', \tau \models F_t\right\}$
else if $\mathrm{type}(t) = \mathrm{rem}$ *and* $a \in \chi_{t'} \setminus \chi_t$ **then**
  $\mathrm{tab}_t \leftarrow \left\{\tau \setminus \{a\} \mid \tau \in \mathrm{tab}''\right\}$
else if $\mathrm{type}(t) = \mathrm{join}$ **then**
  $\mathrm{tab}_t \leftarrow \left\{\tau \mid \tau \in \mathrm{tab}'', \tau \in \mathrm{tab}''\right\}$
return tab_t

Graphical Models

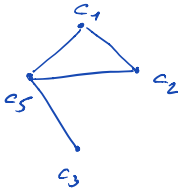
$F = \{c_1, c_2, c_3, c_4, c_5\}$ $\quad c_1 = \{x_1, x_2, x_3\}$ $\quad c_2 = \{\bar{x}_1, x_4\}$ $\quad c_3 = \{x_2, x_5\}$

$c_4 = \{\bar{x}_5, x_6\}$ $\quad c_5 = \{x_1, x_2, \bar{x}_4\}$

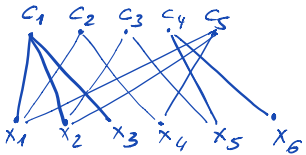
Primal Graph



Dual Graph



Incidence Graph



$inc\text{-}tw(F) \leq primal\text{-}tw(F) + 1$

$inc\text{-}tw(F) \leq dual\text{-}tw(F) + 1$

primal tw and inc-tw - incomp.

(large clause $\Rightarrow$ high prim tw

variable that occ. $\Rightarrow$ high dual tw

in many cls

Outline

3. Finding Tree Decompositions

Empirical Work

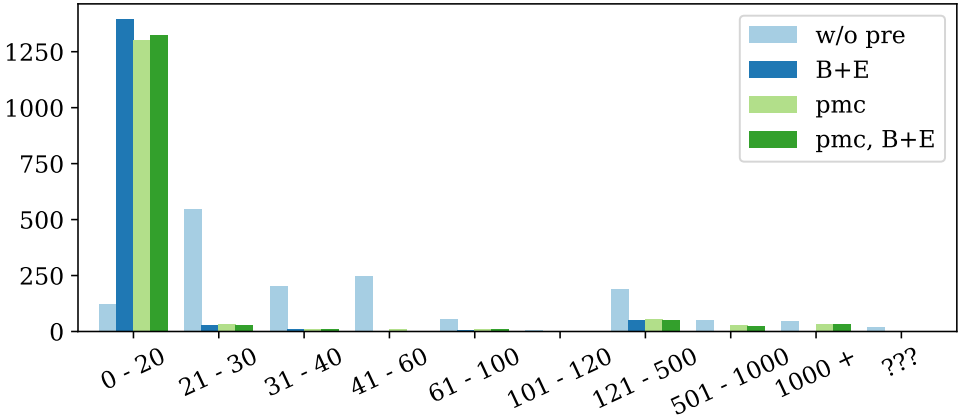
Starting Point

- Research a set of instances and analyze them
- Standard instance set

Disclaimer (following slides)

- Older instance set (2019)
- Slightly different picture on other instances
- 2021 solvers improved
- (lack of resources to rerun frequently)

#SAT: Width Comparison (w/o Preprocessing)



$\Rightarrow$ Produce decompositions of significantly smaller width

Outline

4. Dynamic Programming (GPU)

Parallizing Dynamic Programming

How to parallelize DP?

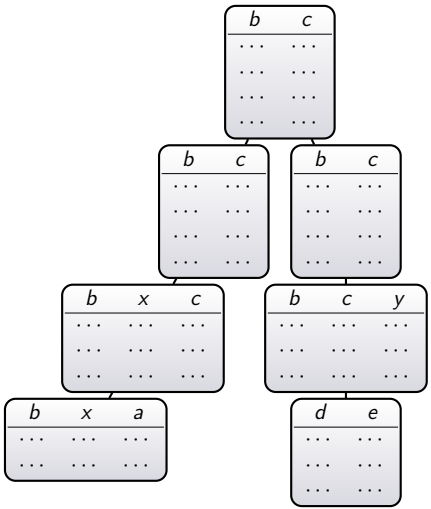
1. Compute tables for multiple nodes in parallel

⇒ 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



Parallizing Dynamic Programming

How to parallelize DP?

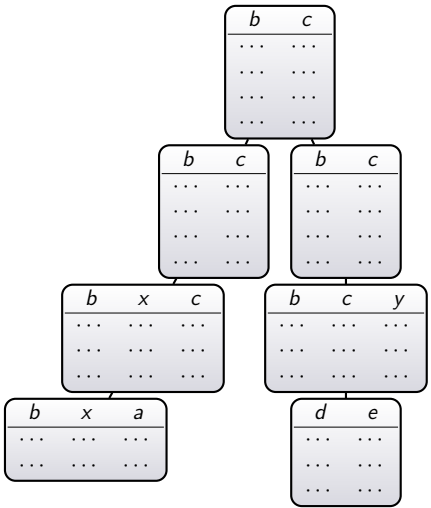
1. Compute tables for multiple nodes in parallel

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

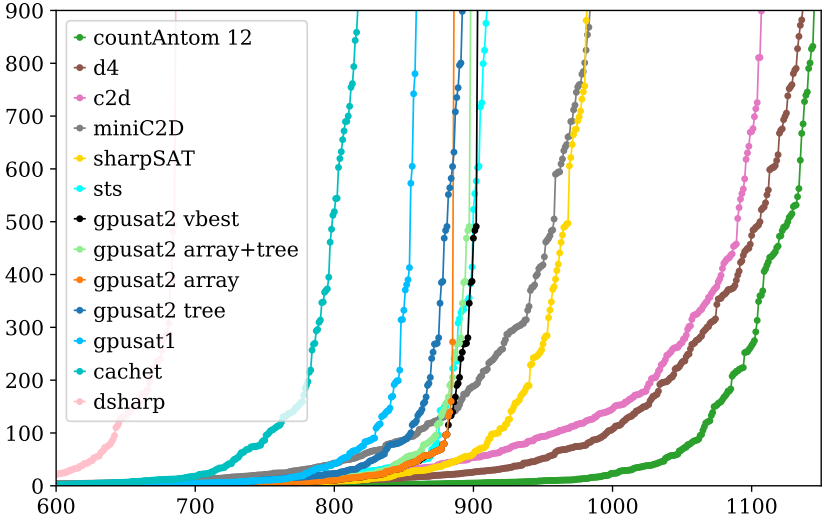
2. Distribute computation of rows among different computation units

⇒ Allows with right hindsight for massive parallelization

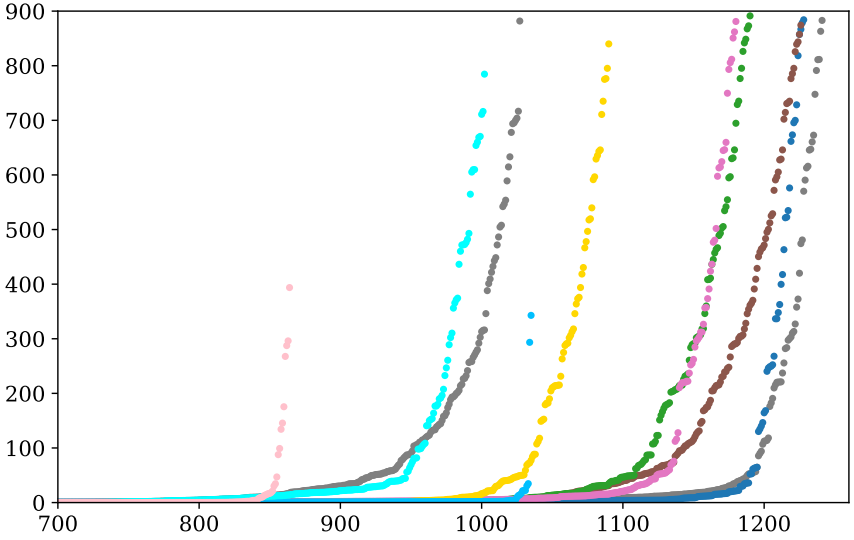
Why: computation of rows are independent



#SAT: Runtime Results (wo. Preprocessing)



#SAT: Runtime Results (w. Preprocessing)



→ Techniques pay off after preprocessing

Outline

5. Dynamic Programming (Databases)

Paralleling Dynamic Programming

How to parallelize DP?

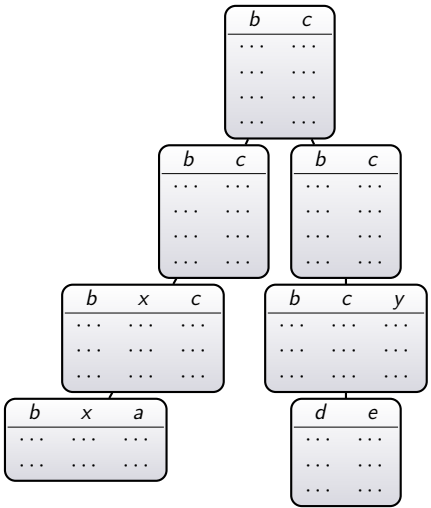
1. Compute tables for multiple nodes in parallel

⇒ 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



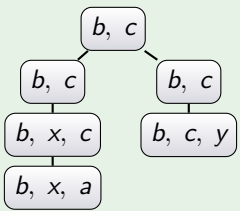
Use a database

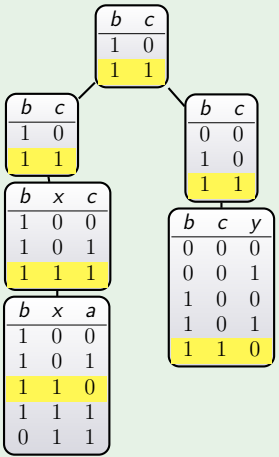
OR how to parallelize nodes and rows?

Solving #SAT [SamerSzeider'10]

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows

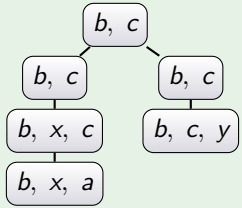




Relational Algebra

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

1. Create graph representation
2. Decompose graph
3. Solve subproblems
4. Combine rows



"Local formula" F_t clauses whose variables are contained in the bag

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** **return** $\begin{pmatrix} cnt \\ 1 \end{pmatrix}$
2 **else if** intr and $a \in V_t$ new **then**
3 **return** $\sigma_{row \models F_t} \left(T_{\downarrow} \times \begin{pmatrix} a \\ 0 \\ 1 \end{pmatrix} \right)$
4 **else if** rem, and $a \notin V_t$ removed **then**
5 **return**

$$\left(\begin{array}{cccc} x_1 & x_2 & \cancel{a} & cnt \\ \hline 0 & 0 & \cancel{0} & c^0_{00} \Rightarrow \sum_{c^0_{00}+c^1_{00}} \\ \cancel{0} & \cancel{0} & \cancel{1} & \cancel{c^1_{00}} \\ 0 & 1 & \cancel{0} & c^0_{01} \Rightarrow \sum_{c^0_{01}+c^1_{01}} \\ \cancel{0} & \cancel{1} & \cancel{1} & \cancel{c^1_{01}} \\ \dots & & & \end{array} \right)$$

6 **else if** join **then**
7 **return** $T_{\downarrow l} \bowtie T_{\downarrow r}$ + "fix cnt (multiply

Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_{\downarrow}$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_{\downarrow}$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_\downarrow$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** **return** $\begin{pmatrix} cnt \\ 1 \end{pmatrix}$
2 **else if** intr and $a \in V_t$ new **then**
3 **return** $\sigma_{row \models F_t}\left(T_\downarrow \bowtie \begin{pmatrix} a \\ 0 \\ 1 \end{pmatrix}\right)$
4 **else if** rem, and $a \notin V_t$ removed **then**
5 **return**

x_1	x_2	~~a~~	~~cnt~~	
0	0	~~0~~	~~c^0_{00}~~	$\Rightarrow \sum_{c^0_{00}+c^1_{00}}$
~~0~~	~~0~~	~~1~~	~~c^1_{00}~~	
0	1	~~0~~	~~c^0_{01}~~	$\Rightarrow \sum_{c^0_{01}+c^1_{01}}$
~~0~~	~~1~~	~~1~~	~~c^1_{01}~~	
...				

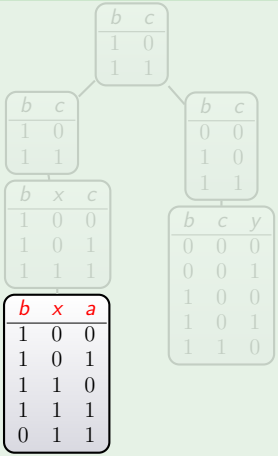
6 **else if** join **then**
7 **return** $T_{\downarrow l} \bowtie T_{\downarrow r}$ + "fix cnt (multiply

Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_\downarrow$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

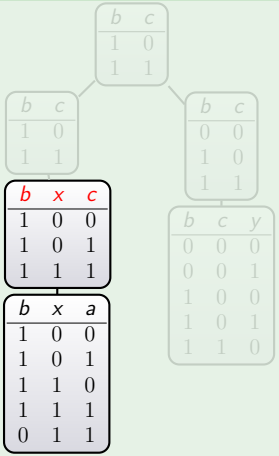


Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_\downarrow$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

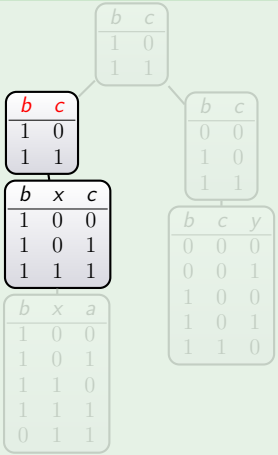


Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_\downarrow$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

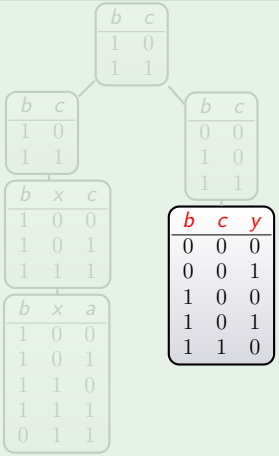


Formulate it in SQL

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

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_{\downarrow}$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`

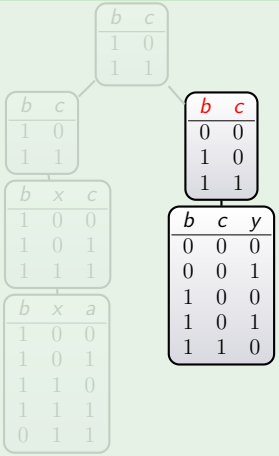


Formulate it in SQL

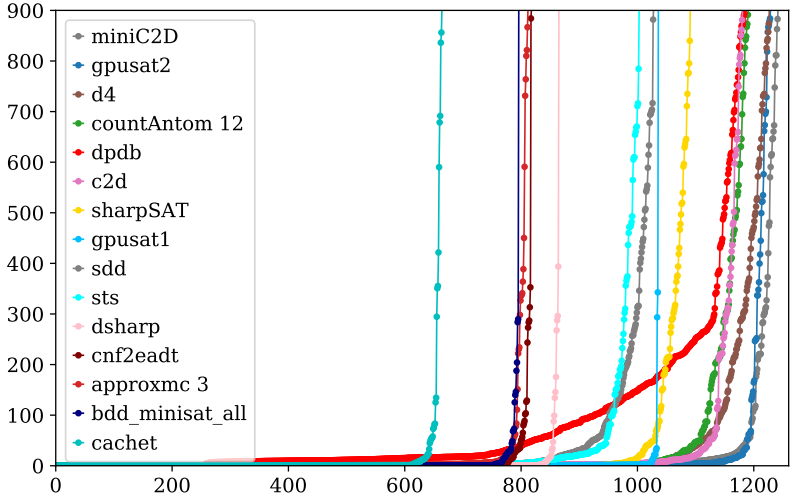
$F = (\neg a \vee b \vee x) \wedge (a \vee b) \wedge (c \vee \neg x) \wedge (b \vee \neg c) \wedge (\neg b \vee \neg c \vee \neg y)$

In: Variables V_t at t, previously computed tables $T_{\downarrow l}$ and $T_{\downarrow r}$
Out: New Table

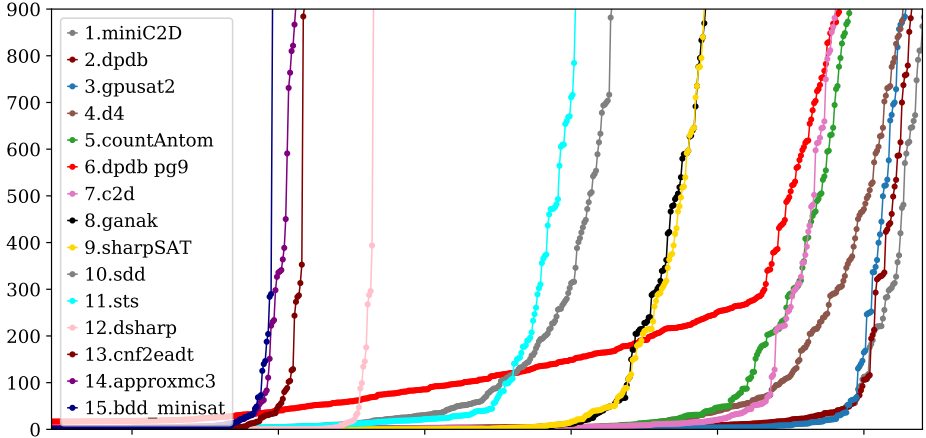
1 **if** leaf **then** `SELECT 1 AS cnt`
2 **else if** intr and $a \in V_t$ new **then**
3 `WITH introduce AS (SELECT 1 AS a UNION ALL SELECT 0) SELECT * FROM` $T_{\downarrow}$`, introduce WHERE (`$l_{1,1}$ `OR ... OR` l_{1,k_1}`) AND ... AND (`$l_{n,1}$ `OR ... OR` l_{n,k_n}`)`
4 **else if** rem, and $a \notin V_t$ removed **then**
5 `SELECT SUM(cnt) AS cnt` + project "a" column using GROUP BY
6 **else if** join **then**
7 `SELECT * from` T_l, T_r `WHERE` $a_{1,l} = a_{1,r}$, $a_{2,l} = a_{2,r}$, ... + `UPDATE cnt` T_l`.cnt * ... *` T_r`.cnt AS cnt`



#SAT (Runtime)


miniC2D
gpusat2
d4
countAntom 12
dpdb
c2d
sharpSAT
gpusat1
sdd
sts
dsharp
cnf2eadt
approxmc 3
bdd_minisat_all
cachet
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200

#SAT (Ram-Disk/PG12)



#SAT (Ram-Disk/PG12)

		solver	0-20	21-30	31-40	41-50	51-60	>60	best	unique	$\sum$	time[h]
preprocessed by pmc	1.	miniC2D	1193	29	**10**	2	1	7	11	0	**1242**	**68.77**
	2.	dpdb	1193	31	7	2	0	0	2	1	1233	70.44
	3.	gpusat2	**1196**	**32**	1	0	0	0	154	1	1229	71.27
	4.	d4	1163	20	**10**	2	**4**	28	33	1	1227	76.86
	5.	countAntom	1141	18	**10**	5	**4**	13	102	0	1191	84.39
	6.	dpdb pg9	1159	19	5	2	0	0	0	0	1184	100.99
	7.	c2d	1124	31	**10**	3	3	10	24	0	1181	84.41
	8.	ganak	1031	16	10	2	**4**	29	**633**	0	1092	107.25
	9.	sharpSAT	1029	16	**10**	2	**4**	30	80	0	1091	106.88
	10.	sdd	1014	4	7	1	0	2	0	0	1028	124.23
	11.	sts	927	4	8	**7**	5	**52**	35	**21**	1003	128.43
	12.	dsharp	853	3	7	2	0	0	29	0	865	157.87
	13.	cnf2eadt	799	3	7	2	0	7	157	0	818	170.17
	14.	approxmc 3	794	3	7	2	0	6	1	0	812	173.35
	15.	bdd_minisat	791	4	1	0	0	0	46	0	796	175.09

Lessons Learned: GPU

Implementation

- You need to get your hands dirty
- Architecture matters (first OpenCL / latest version CUDA)
- Implementation for inc-tw rarely pays off
  (inctw+1 <= primtw; can be exponentially smaller)

"Constants Matter"

- Boils sometimes down to bit fiddling
- Use low level data structures
- Avoid copying (RAM2VRAM slow)
- Efficient heuristics are key

$\Rightarrow$ Works surprisingly well

Lessons Learned: Databases

Implementation

- Works surprisingly well (gives a framework for many problems)
- Has the obvious theoretical limitations
- Again: details matter
- Difference between storing data in memory and on the disk large (just a constant)
- Don't try to compete with years of engineering without need (algorithms inside databases)
- Treewidth might not be enough

Summary

Contributions

- Python framework to implement DPs
- Use SQL to specify algorithm
- Exploit modern database systems
- Framework is surprisingly competitive

Benchmark: Comparing apples and oranges

Maybe...

Implementation

Disclaimer for theorists: you need practical experiences with databases

+

Right hindsight

Implementation

Right hindsight?

1. Use the database as much as possible
2. Use Views instead of actual storage (so the database can handle it in memory)
3. Allow parallel processing (24 worker threads)

Actual Implementation

1. Python 3.5 (main API)
2. PostgreSQL 9.6

Show off (1518 lines of python code)

1. SAT Solver 72 lines
2. #SAT Solver 94 lines
3. Vertex Cover 199 lines

How does the Python code look like?

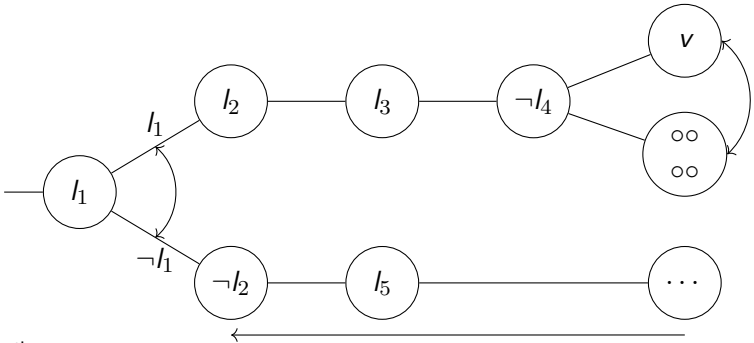
```
class SharpSat(Problem):
...
    def td_node_column_def(self,var):
        return td_node_column_def(var)
    def td_node_extra_columns(self):
        return [("model_count","NUMERIC")]
    def candidate_extra_cols(self,node):
        return ...
    def assignment_extra_cols(self,node):
        return ["sum(model_count) AS model_count"]
    def filter(self,node):
        return filter(self.var_clause_dict, node)
    def prepare_input(self, fname):
        input = CnfReader.from_file(fname)
        ...
        return cnf2primal(input.num_vars, input.clauses,
           self.var_clause_dict)
    def setup_extra(self):
        create_tables()
        insert_data()
    def after_solve(self):
        ...
```

Outline

C) Modern Solving Techniques

Modern Solving Techniques: Component Caching

- Systematic Search Space Splitting + CDCL
- Split formula systematically by assigning variables
- Component of formula F set C where $\mathrm{vars}(C) \cap \mathrm{vars}(F - C) = \emptyset$
- Store values for computed components

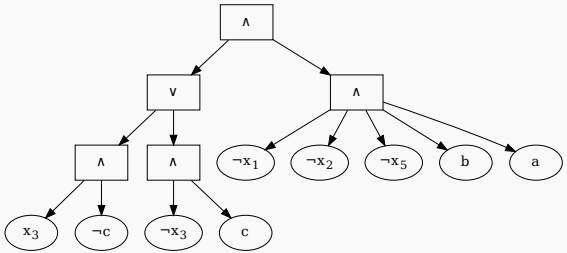


Modern Solving Techniques: Knowledge Compilation

- Transform CNF Formula into sd-DNNF; counting on DAG representation
- NNF: neg. directly in front of variables, conj., disj.
- D (decomposable): sides of the conj. don't share vars
- d (deterministic): Disjuncts are logically disjoint
- s (mooth): $\psi = \psi_1 \vee \psi_2$, $\mathsf{vars}(\psi_1) = \mathsf{vars}(\psi_2)$

Example:

$F_1 =$
$((x_3 \wedge \neg c) \vee (\neg x_3 \wedge c)) \wedge$
$(\neg x_1 \wedge \neg x_2 \wedge \neg x_5 \wedge a \wedge b)$

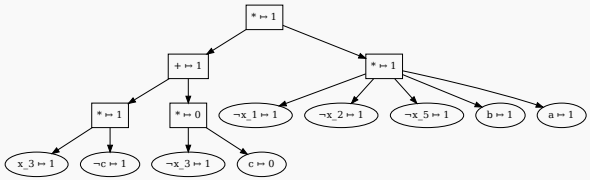


Modern Solving Techniques: Knowledge Compilation

- Transform CNF Formula into sd-DNNF; counting on DAG representation
- NNF: neg. directly in front of variables, conj., disj.
- D (decomposable): sides of the conj. don't share vars
- d (deterministic): Disjuncts are logically disjoint
- s (mooth): $\psi = \psi_1 \vee \psi_2$, $\mathsf{vars}(\psi_1) = \mathsf{vars}(\psi_2)$

Example:

$F_1 =$
$((x_3 \wedge \neg c) \vee (\neg x_3 \wedge c)) \wedge$
$(\neg x_1 \wedge \neg x_2 \wedge \neg x_5 \wedge a \wedge b)$



Certifying the Results of Model Counters

Proofs

- Solvers make mistakes, change constantly
- Solvers cannot be verified entirely $\Rightarrow$ trace output and certify the results
- Define proof system based on basic search space splitting techniques
- Reuse existing SAT proof techniques (DRAT)
- Building blocks: components/assumptions
  exactly one model, compose, join, extend rules

Experimental Work (Runtime Disclaimer)

Questionable Setting?

Aren't you comparing apples and oranges? YES.

Problems of the Setting

- We compare on different hardware
- Wallclock is unfair

  Usually user is interested in getting things done quickly (+ fairly cheap)

$\Rightarrow$ Power consumption (Joule) and price of investment better measure
(BUT not accessible with the current framework)

- Parallel vs. sequential: No excuse, sorry

Problem of Interest (cont.)

Weighted Model Counting (WMC/Sum of Product)

- 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/Sum of Product)

- 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.,

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

Experimental Work

Instances

- 1400 instances from public benchmarks

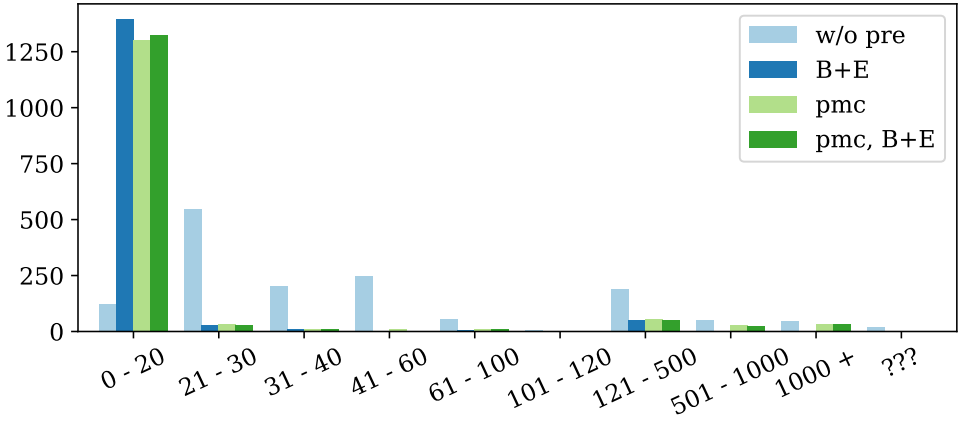
Limits

- Cannot expect to solve instances of high treewidth

Setting (Runtime Comparison)

- Take recent solvers.
- Consider Wallclock

#SAT: Width FHZisser'19



Experiments (Setup)

Hardware

- non-GPU solving: cluster of 9 nodes; each two E5-2650 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

Algorithm for Primal Graph

In: Node t, bag χ_t, clauses F_t, sequence C of tables.
Out: Table tab_t

if $\mathrm{type}(t) = \mathrm{leaf}$ **then**
  $\mathrm{tab}_t \leftarrow \{\emptyset\}$
else if $\mathrm{type}(t) = \mathrm{intr}$ *and* $a \in \chi_t \setminus \chi_{t'}$, **then**
  $\mathrm{tab}_t \leftarrow \left\{\tau \cup \{a\} \mid \tau \in \mathrm{tab}'', \tau \cup \{a\} \models F_t\right\} \cup$
  $\left\{\tau \mid \tau \in \mathrm{tab}'', \tau \models F_t\right\}$
else if $\mathrm{type}(t) = \mathrm{rem}$ *and* $a \in \chi_{t'} \setminus \chi_t$ **then**
  $\mathrm{tab}_t \leftarrow \left\{\tau \setminus \{a\} \mid \tau \in \mathrm{tab}''\right\}$
else if $\mathrm{type}(t) = \mathrm{join}$ **then**
  $\mathrm{tab}_t \leftarrow \left\{\tau \mid \tau \in \mathrm{tab}'', \tau \in \mathrm{tab}''\right\}$
return tab_t

Backup Slides

Experimental Work (Runtime Disclaimer)

Questionable Setting?

Aren't you comparing apples and oranges? YES.

Problems of the Setting

- We compare on different hardware
- Wallclock is unfair

  Usually user is interested in getting things done quickly (+ fairly cheap)

$\Rightarrow$ Power consumption (Joule) and price of investment better measure
(BUT not accessible with the current framework)

- Parallel vs. sequential: No excuse, sorry

Problem of Interest (cont.)

Weighted Model Counting (WMC/Sum of Product)

- 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/Sum of Product)

- 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.,

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

Summary

Contributions

- Python framework to implement DPs
- Use SQL to specify algorithm
- Exploit modern database systems
- Framework is surprisingly competitive

Benchmark: Comparing apples and oranges

Maybe...

Implementation

Disclaimer for theorists: you need practical experiences with databases

+

Right hindsight

Implementation

Right hindsight?

1. Use the database as much as possible
2. Use Views instead of actual storage (so the database can handle it in memory)
3. Allow parallel processing (24 worker threads)

Actual Implementation

1. Python 3.5 (main API)
2. PostgreSQL 9.6

Show off (1518 lines of python code)

1. SAT Solver 72 lines
2. #SAT Solver 94 lines
3. Vertex Cover 199 lines

How does the Python code look like?

```
class SharpSat(Problem):
...
    def td_node_column_def(self,var):
        return td_node_column_def(var)
    def td_node_extra_columns(self):
        return [("model_count","NUMERIC")]
    def candidate_extra_cols(self,node):
        return ...
    def assignment_extra_cols(self,node):
        return ["sum(model_count) AS model_count"]
    def filter(self,node):
        return filter(self.var_clause_dict, node)
    def prepare_input(self, fname):
        input = CnfReader.from_file(fname)
        ...
        return cnf2primal(input.num_vars, input.clauses,
           self.var_clause_dict)
    def setup_extra(self):
        create_tables()
        insert_data()
    def after_solve(self):
        ...
```

Experimental Work

Instances

- 1400 instances from public benchmarks

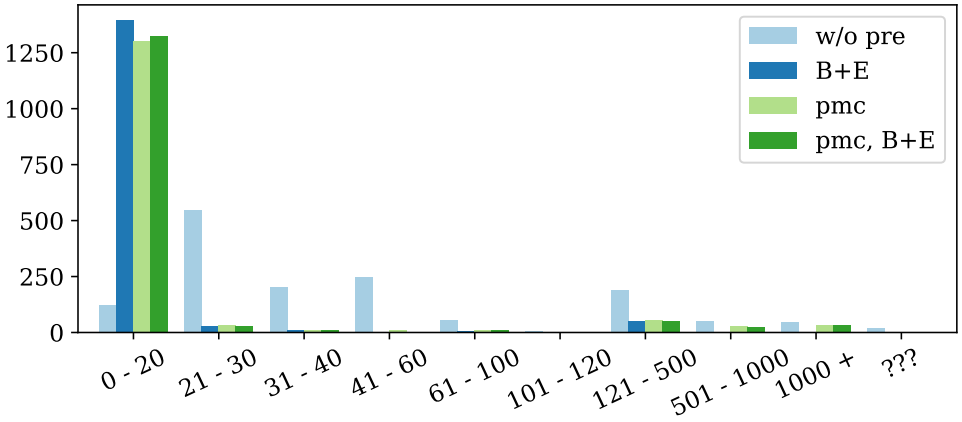
Limits

- Cannot expect to solve instances of high treewidth

Setting (Runtime Comparison)

- Take recent solvers.
- Consider Wallclock

#SAT: Width FHZisser'19



Experiments (Setup)

Hardware

- non-GPU solving: cluster of 9 nodes; each two E5-2650 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

Modern Solving Techniques: Component Caching (cont.)

Cache Cleaning Strategies

1. Cachet (Age-Based) Old entries are useless
   - **Metric:** Age (time since creation).
   - **Trigger:** When cache is full (e.g., 2^{21} entries).
   - **Action:** Delete the oldest entries (keep 25% newest).
2. SharpSAT (Activity-Based)
   - **Metric:** Score (Activity). Used entries are useful.
     - Init: Age.
     - Hit: Reset score (boost).
     - Periodic: Decay all scores.
   - **Trigger:** Cache exceeds max size fraction.
   - **Action:** Delete lowest scores.

Adapting to Counting: VSADS

VSADS (Sang et al., 2004)

Variable State Aware Decaying Sum
A hybrid score combining activity and structural relevance:

$$\text{Score}(v) = \alpha \cdot \textbf{Activity}(v) + \beta \cdot \textbf{Count}(v)$$

- **Activity:** Standard VSIDS score (conflicts).
- **Count:** How many times v appears in the current formula.
- **Dynamic:** If many conflicts occur, α dominates (act like CDCL). If formula simplifies, β dominates (act like MOMS/Decomposition).

Cache-Aware Branching: CSVSADS

- **Component Caching** is the primary speedup mechanism in counting
- Branch on variables that define cache keys, increase probability of a **Cache Hit**.

CSVSADS (Sharma et al., 2019b)

Cache Score based VSADS

$$\text{Score}(v) = \text{VSIDS}(v) + \textbf{CacheScore}(v)$$

How CacheScore works:

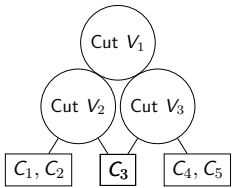
- When storing a component in the cache, increase score of variables appearing in that component.
- **Intuition:** If a variable appears in many cached components, assigning it might lead to a state we have seen before.
- Reduces the number of unique components generated during search.

Structure-Guided Branching: The Static Approach (C2D)

The Idea (Darwiche, 2004)

Instead of letting the solver wander (VSIDS), we force it to follow a **pre-computed roadmap** that guarantees decomposition.

1. **Offline Phase:** Construct a **dtree** (Decomposition Tree) for the CNF.
   - A binary tree where leaves are clauses.
   - Internal nodes represent the union of clauses.
2. **Online Phase (Compilation):**
   - Visit the dtree nodes.
   - Branch on the **cutset** variables (variables shared between the left and right child).



Variables in "Cut" are instantiated first to separate the clauses below.

- **Pros:** Guarantees low treewidth behavior.
- **Cons: Static.** Does not adapt to the simplifications (unit propagations) occurring during the search.

Dynamic Structural Branching (D4)

The idea (Lagniez and Marquis, 2017)

D4 integrates graph partitioning directly into the search loop:

1. **Compute Cutset:** Call a partitioner (e.g., KaHyPar) to find a separator S for the current component.
2. **Filter:** Restrict branching candidates to S (ignoring others).
3. **Select:** Choose the best variable $v \in S$ using VSADS.
4. **Repeat:** When S is exhausted, re-partition the remaining graph.

Advantage

Computes cuts on the **simplified** graph (post-propagation). Often yields much smaller separators than static methods (C2D).

Drawbacks

- **Overhead:** Partitioning at runtime is expensive.
- **"Tunnel Vision":** Ignores high-activity variables outside S, potentially delaying conflict resolution.

Structure-Guided Branching (sharpSAT-TD)

Concept: Hybrid Guidance

(Korhonen and Järvisalo, 2021)

Theory suggests counting is efficient for small treewidth k ($O(2^k n)$), but practical k is often too large. **sharpSAT-TD** uses Tree Decomposition (TD) as a **soft guide** rather than a rigid order.

Modified Heuristic

$$\mathrm{sc}(x) = \underbrace{\mathrm{act}(x) + \mathrm{fr}(x)}_{\text{VSADS}} - \underbrace{C \cdot \min_{t \in T_x} d_T(t)}_{\text{Struct. Penalty}}$$

Key Components:

- **Penalty:** $d_T(t) \in [0, 1]$ is the distance of bag t from the root.
- **Effect:** Variables at the "top" of the TD (separators) have lower penalty $\rightarrow$ higher priority.

- **Tuning:** C scales with width w.
  - **Large** C**:** Strictly follows decomposition order.
  - **Small** C**:** Behaves like standard conflict-driven search.