Parameterized Algorithmics and Counting: Treewidth in Practice

Johannes K. Fichte

TU Wien

PCCR @FLoC, August 1st, 2022

Preliminaries: Problem of Interest

SAT-Problem (Boolean Satisfiability Problem)

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

Example

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

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

Model Counting (#SAT/Number SAT)

- Number of satisfying truth assignments to F.

Motivation

Qualitative (Decision or Optimization)

- Does the problem have a solution?
- Output a reason for a decision.

Applications

- Various applications in AI and reasoning
  - Bayesian reasoning [Sang et al.'05]
  - Learning distributions [Choi et al.'15]
  - Infrastructure reliability [Meel et al.17]

Quantitative Questions

- How many solutions does the problem have?
- How likely is an occurrence?

Computational complexity

- #P-complete [Valiant'79]

Motivation: Related Works.

#SAT Solving

- Various solvers (Model Counting Competition 2020):
  - approximate [Meel et al.]
  - component caching [Baccus/Thurley/Meel & Soos]
  - knowledge compilation based [Choi,Darwiche/Lagniez,Marquis et al.]

Theory: fast on instances of low primal treewidth [Baccus, Dalmao, Pitassi'03]

Why still interesting?

- Unknown for some parameters [Stefan's talk]
- Modern hardware massively parallel

$\Rightarrow$ Some parameterized algorithms allow for parallelization

Motivation: Related Works.

#SAT Solving

- Various solvers (Model Counting Competition 2020):
  - approximate [Meel et al.]
  - component caching [Baccus/Thurley/Meel & Soos]
  - knowledge compilation based [Choi,Darwiche/Lagniez,Marquis et al.]

Theory: fast on instances of low primal treewidth [Baccus, Dalmao, Pitassi'03]

Why still interesting?

- Unknown for some parameters [Stefan's talk]
- Modern hardware massively parallel

$\Rightarrow$ Some parameterized algorithms allow for parallelization

Motivation: Related Works.

#SAT Solving

- Various solvers (Model Counting Competition 2020):
  - approximate [Meel et al.]
  - component caching [Baccus/Thurley/Meel & Soos]
  - knowledge compilation based [Choi,Darwiche/Lagniez,Marquis et al.]

Theory: fast on instances of low primal treewidth [Baccus, Dalmao, Pitassi'03]

Why still interesting?

- Unknown for some parameters [Stefan's talk]
- Modern hardware massively parallel

$\Rightarrow$ Some parameterized algorithms allow for parallelization

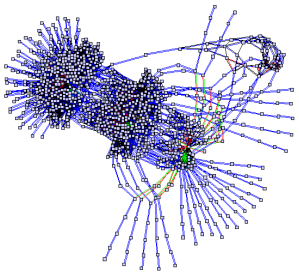
Motivation: Background

Theory:

SAT hard to solve! (Pick your favorite conjecture: ETH, SETH...)

Idea:

1) Practical instances are usually highly structured
2) Structure can be exploited by algorithms



i) Solvers exploit structure (don't necessarily know how)
iia) Proof theory: understanding on possibilities and limitations
iib) Parameterized algorithmics: solve special cases efficiently
(Size + ... We don't talk about just the number of variables.)

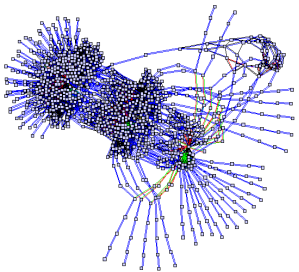
Motivation: Background

Theory:

SAT hard to solve! (Pick your favorite conjecture: ETH, SETH...)

Idea:

1) Practical instances are usually highly structured
2) Structure can be exploited by algorithms



i) Solvers exploit structure (don't necessarily know how)
iia) Proof theory: understanding on possibilities and limitations
iib) Parameterized algorithmics: solve special cases efficiently
(Size + ... We don't talk about just the number of variables.)

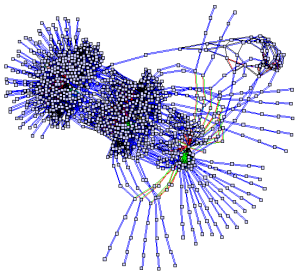
Motivation: Background

Theory:

SAT hard to solve! (Pick your favorite conjecture: ETH, SETH...)

Idea:

1) Practical instances are usually highly structured
2) Structure can be exploited by algorithms



i) Solvers exploit structure (don't necessarily know how)
iia) Proof theory: understanding on possibilities and limitations
iib) Parameterized algorithmics: solve special cases efficiently
(Size + ... We don't talk about just the number of variables.)

Motivation: Parameterized Algorithms

Last 20 years

- Lots a theoretical work and various algorithms for #SAT
- Quite a lack of implementing and trying those algorithms

Research Question

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

Motivation: Parameterized Algorithms

Last 20 years

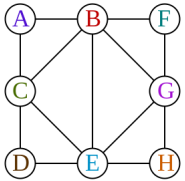
- Lots a theoretical work and various algorithms for #SAT
- Quite a lack of implementing and trying those algorithms

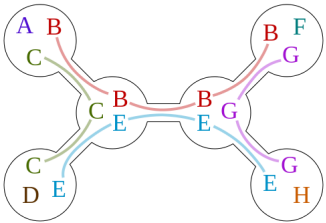
Research Question

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

Parameterize: Decompositions

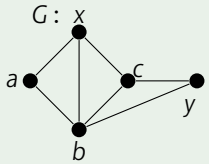
- Idea: decompose the problem into subproblems, and combine solutions to subproblems to a global solution
- Parameter: overlap between subproblems
- Treewidth of the primal graph

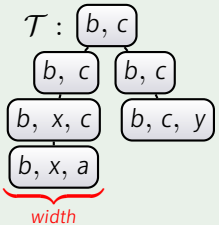




Tree Decompositions

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

Topic of the Remaining Talk

Implementations

“Find” tree decompositions of small width? (Heuristic, not treewidth; otherwise, NP-c)

Works well even for relatively large instances.

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

“Find” tree decompositions of small width? (Heuristic, not treewidth; otherwise, NP-c)

Works well even for relatively large instances.

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

Empirical Work

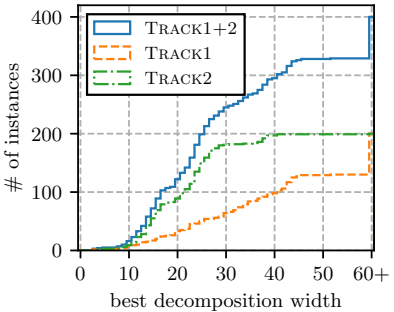
Instances

- Past: 2585 instances from public benchmarks [FHecherWoltranZisser'18,'19]

$\Rightarrow$ 54% primal treewidth below 30; 70% below 40 (MinFill+MinDegree)

- MCC2020-Track1+2 [FHecherRoland'21] 400 instances

$\Rightarrow$

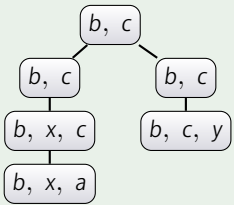


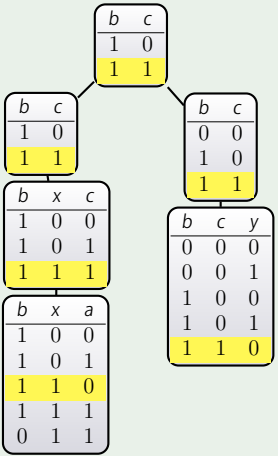
How to “use” tree decompositions for #SAT?

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





Paralleling Dynamic Programming

How to parallelize DP?

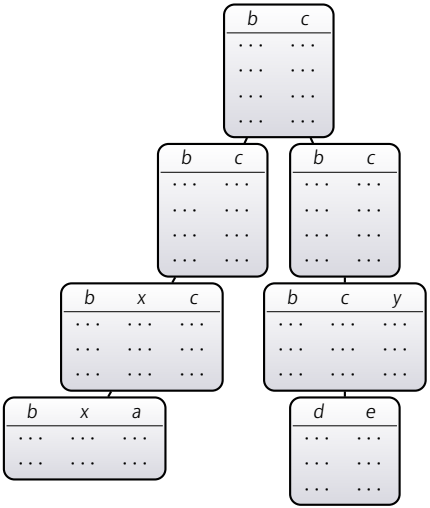
1. Compute tables for multiple nodes in parallel

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

2. Distribute computation of rows among different computation units

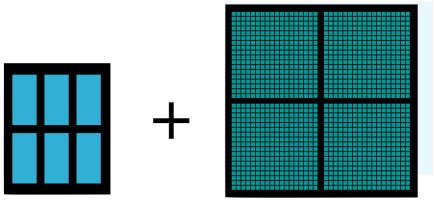
⇒ Allows with right hindsight for massive parallelization

Why: computation of rows are independent



A GPU-based #SAT-solver

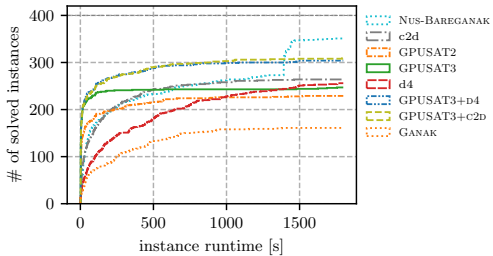
OR how to go massively parallel?

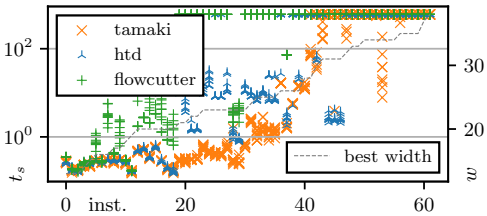


Empirical Comparison

(Model counters improved significantly over the last two years)

Outcome: Runtime / Treewidth





Outcome: More Optimistic Picture

Solver	# inst.	∑t 100%	∑t 95%	∑t 90%	∑t 50%
GPUSAT2	229	5:56:06	3:05:47	1:51:28	0:09:13
...on baseline	175	2:26:36	1:00:06	0:36:32	0:06:38
GPUSAT3	247	**3:05:58**	**0:43:40**	**0:27:12**	**0:07:01**
...on baseline	175	**0:18:08**	**0:13:00**	**0:11:03**	**0:04:51**
D4	256	1 day, 2:30:28	20:57:09	16:39:14	1:59:09
...on baseline	175	15:54:58	12:09:37	9:21:47	0:53:41
C2D	265	12:25:56	8:20:19	6:21:38	0:39:15
...on baseline	175	3:29:07	2:16:12	1:40:38	0:13:16
NUS-BAREGANAK	**351***	1 day, 21:47:59	1 day, 14:21:25	1 day, 7:41:05	1:33:55
...on baseline	175	3:12:16	1:57:25	1:30:17	0:17:53
GANAK	161	11:26:48	9:05:15	7:28:31	0:53:24
GPUSAT3+D4	304	7:36:44	3:36:43	1:58:31	0:09:05
...on baseline	175	**0:18:08**	**0:13:00**	**0:11:03**	**0:04:51**
GPUSAT3+C2D	309	8:45:15	4:30:15	2:35:57	0:09:23
...on baseline	175	**0:18:08**	**0:13:00**	**0:11:03**	**0:04:51**

Lessons Learned?

Implementation

- You need to get your hands dirty
- Architecture matters
- Implementation for inc-tw rarely paid off; dual-tw never paid off

"Constants Matter"

- Bit fiddling
- Low level data structures
- Avoid copying (VRAM-RAM transfer slow)
- Efficient heuristics and preprocessing key

$\Rightarrow$ Works surprisingly well

Use a database

OR how to parallelize nodes and rows?

System DPDB

Idea

- Avoid implementing fiddling details of complicated dynamic programming algorithms (similar algorithms for various problems in AI)
- Just describe main parts in relational algebra (SQL)
- Employ modern database systems and years of engineering in algorithms on tables and heuristics
- You can adapt the degree of parallelity (even sequential) by adapting the database config

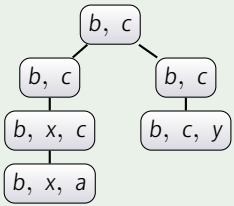
Implementation

- Works surprisingly well (gives a framework for many problems), but details matter
- Just difference between storing data in memory and on the disk is massive (just a constant)
- Don't compete with years of engineering without need (database algorithms)
- Treewidth might not be enough

Dynamic Programming: Relational Algebra/SQL

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

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)

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`

Outcome

DPDB [FHecherWoltranThier'20]

- Competitive with preprocessing
- Show off (1518 lines of python code):
- SAT Solver 72 lines; #SAT Solver 94 lines; Vertex Cover 199 lines

NestHDB [Hecher et al.'20]

- Use approach together with SAT solvers locally
- Abstract from treewidth; hybrid solving

$\Rightarrow$ Solve projected model counting (#*NP*) competitively

$\Rightarrow$ Instances where Primal Tw is 200+

Summary

Applications (Practical Framework for Dynamic Programming)

1. Abstract Argumentation [Dewoprabowo et al.'22]
2. Answer Set Programming + Probabilistic Reasoning [FHecherNadeem'22]
3. Epistemic Logic Programming [Hecher et al.'21]
4. ...
5. Competitive for Decision Problems? $\Rightarrow$ not likely

Today's best solver: SharpSAT-tw

- Tree decompositions as heuristic for search space splitting [Korhonen& Jarvisalo'21]

Conclusion

Take Home Messages

1. Parameterized Algorithms can work
2. Download at **github.com/daajoe/gpusat** or **github.com/hmarkus/dpondbs**
3. PACE'16&'17 influenced other communities

Future

- Parallel model counting & algorithmic considerations
- PACE again with treewidth?

Thanks for listening! & Questions?
Advertisement: Markus' Talk in QBF workshop (4pm)

Collaborators:

Arne Meier, Dominik Rustovac, Markus Hecher, Markus Zisser, Patrick Thier, Sarah Gaggl, Stefan Woltran, Valentin Roland, ...