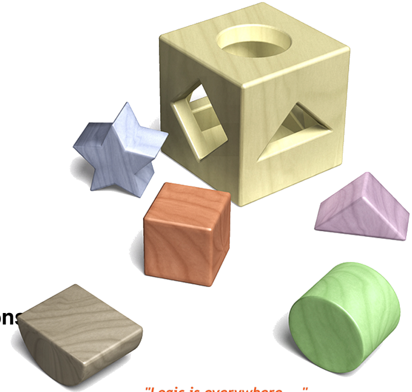
Visualizing Dynamic Programming On Tree Decompositions

Martin Röbke
Fakultät Informatik
Technische Universität Dresden
Germany

- **REWORK AFTER CHAPTERS**
- WHAT **was the motivation**
- WHAT **could be used otherwise?**
- WHO **benefits from visualization?**
- METHODOLOGY **challenges and solutions**
- WHAT **could be developed next?**



"Logic is everywhere ..."

Motivation

- DP-on-TD-algorithms can solve Model Counting and various combinatorial problems
- Implementations of those are competing with modern solvers
- **But:** those are fairly hard to implement efficiently
- Practical debug output quickly becomes very large (GB)
- Finding the cause of the problem is a time consuming challenge

Background

The algorithms of interest solve problems of:

- **combinatorics (NP-problems)**
- **model-counting (#P-problems)**

Recent promising results for Projected Model Counting by Markus Hecher[1].

[1] Hecher M., Thier P., Woltran S. (2020) Taming High Treewidth with Abstraction, Nested Dynamic Programming, and Database Technology. In: Pulina L., Seidl M. (eds) Theory and Applications of Satisfiability Testing - SAT 2020. SAT 2020. Lecture Notes in Computer Science, vol 12178. Springer, Cham. https://doi.org/10.1007/978-3-030-51825-7_25

Tree Decompositions

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

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

Graphic

Graphs for Boolean Formulas

- **Example set of CNF-clauses:**

  {c1 = {v1, v3, ¬v4}, c2 = {¬v1, v6}, c3 = {¬v2, ¬v3, ¬v4}, c4 = {¬v2, v6}, c5 = {¬v3, ¬v4}, c6 = {¬v3, v5}, c7 = {¬v5, ¬v6}, c8 = {v5, v7}}

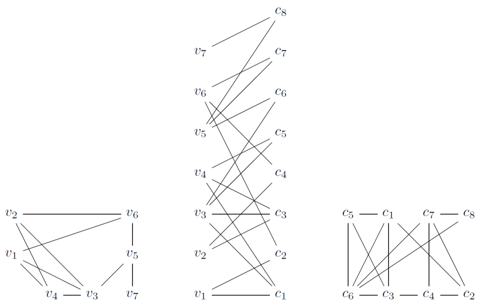
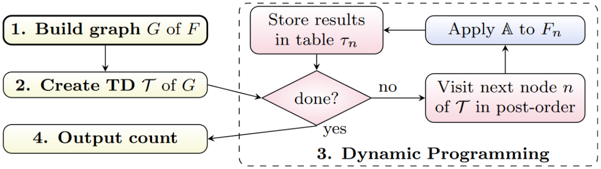
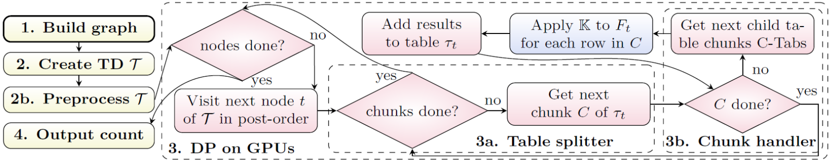


Figure: The primal (left), incidence (middle) and dual (right) graph

gpusat2 - Solving on GPU


1. Build graph G of F
2. Create TD $\mathcal{T}$ of G
4. Output count
Store results in table τ_n
Apply $\mathbb{A}$ to F_n
done?
no
yes
Visit next node n of $\mathcal{T}$ in post-order
3. Dynamic Programming

- **Customized tree decompositions**
- **Adapted memory-management**
- **Improved precision handling**


1. Build graph
2. Create TD $\mathcal{T}$
2b. Preprocess $\mathcal{T}$
4. Output count
nodes done?
no
yes
Visit next node t of $\mathcal{T}$ in post-order
3. DP on GPUs
Add results to table τ_t
Apply $\mathbb{K}$ to F_t for each row in C
Get next child table chunks C-Tabs
yes
chunks done?
no
Get next chunk C of τ_t
3a. Table splitter
no
C done?
yes
3b. Chunk handler

[1] Images: Markus Zisser. *Solving the SAT problem on the GPU with dynamic programming and OpenCL*. Technische Universität Wien, 2018.

dpdb

Using databases for intermediate results

1. **Create graph representation**
2. **Decompose graph**
3. **Solve sub-problems**
4. **Combine rows**

Generator SQL Qs =¿ Datenbank Templating in Python

- **SAT**
- **#SAT**
- **Vertex cover**

– #εTab#: `SELECT 1 AS cnt`
– #intrTab#: `SELECT 1 AS val UNION ALL 0`
– #localProbFilter#: $(l_{1,1}$ `OR` … `OR` $l_{1,k_1})$ `AND` … `AND` $(l_{n,1}$ `OR` … `OR` $l_{n,k_n})$
– #aggrExp#: `SUM(cnt) AS cnt`
– #extProj#: τ_1`.cnt * … *` τ_ℓ`.cnt AS cnt`

(a) Problem #SAT

– #εTab#: `SELECT 1 AS cnt`
– #intrTab#: `SELECT 1 AS val UNION ALL … UNION ALL` o
– #localProbFilter#: `NOT` $([\![u_1]\!] = [\![v_1]\!])$ `AND` … `AND NOT` $([\![u_n]\!] = [\![v_n]\!])$
– #aggrExp#: `SUM(cnt) AS cnt`
– #extProj#: τ_1`.cnt * … *` τ_ℓ`.cnt AS cnt`

(b) Problem #o-Col

– #εTab#: `SELECT 0 AS card`
– #intrTab#: `SELECT 1 AS val UNION ALL 0`
– #localProbFilter#: $([\![u_1]\!]$ `OR` $[\![v_1]\!])$ `AND` … `AND` $([\![u_n]\!]$ `OR` $[\![v_n]\!])$
– #aggrExp#: `MIN(card) AS card`
– #extProj#: τ_1`.card + … +` τ_ℓ`.card -` $(\Sigma_{i=1}^{\ell}|\chi(t_i) \cap \{a_1\}|$ `- 1) *` $\tau_1.[\![a_1]\!]$ `- … -` $(\Sigma_{i=1}^{\ell}|\chi(t_i) \cap \{a_k\}|$ `- 1) *` $\tau_1.[\![a_k]\!]$

(c) Problem MinVC

github: `https://github.com/hmarkus/dp_on_dbs`

Challenge1

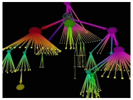
Generisches Datenformat /Strings Visu
=¿ Anwendungen

Challenge2

=¿ Wie robust ist die Datenverarbeitung in der Visu =¿ Was Gedanken bei der Visu waren

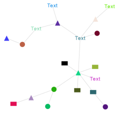
Challenge3

© Gephi.org - a tool for data analysts and scientists keen to explore and understand graphs.[2]



Tulip - Better Visualization Through Research. [3]

[4] Vis.js Sigma.js



vasturiano/3d-force-graph [5]

=¿ Related Work Schluss / Wiss Arbeiten -¿ Nicht speziell Angeschaut / Format aus Solvern extrahiert - kann trotzdem sehr generisch sein (dpdb speziell)

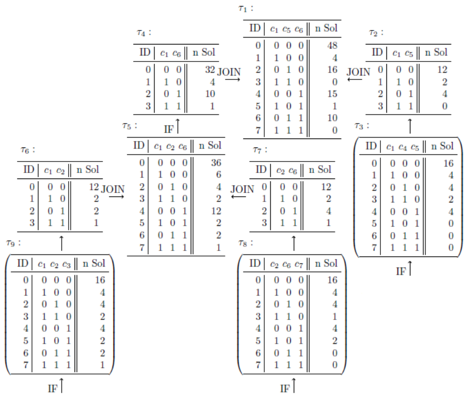
[2] https://gephi.org/

[3] https://tulip.labri.fr/TulipDrupal/

[4] https://neo4j.com/developer/tools-graph-visualization/

Visualization

Manually for gpusat



τ_1 :

ID	c_1	c_5	c_6	n Sol
0	0	0	0	48
1	1	0	0	4
2	0	1	0	16
3	1	1	0	0
4	0	0	1	15
5	1	0	1	1
6	0	1	1	10
7	1	1	1	0

τ_2 :

ID	c_1	c_5	n Sol
0	0	0	12
1	1	0	2
2	0	1	4
3	1	1	0

τ_3 :

ID	c_1	c_4	c_5	n Sol
0	0	0	0	16
1	1	0	0	4
2	0	1	0	4
3	1	1	0	2
4	0	0	1	4
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0

τ_4 :

ID	c_1	c_6	n Sol
0	0	0	32
1	1	0	4
2	0	1	10
3	1	1	1

τ_5 :

ID	c_1	c_2	c_6	n Sol
0	0	0	0	36
1	1	0	0	6
2	0	1	0	4
3	1	1	0	2
4	0	0	1	12
5	1	0	1	2
6	0	1	1	2
7	1	1	1	1

τ_6 :

ID	c_1	c_2	n Sol
0	0	0	12
1	1	0	2
2	0	1	2
3	1	1	1

τ_7 :

ID	c_2	c_6	n Sol
0	0	0	12
1	1	0	2
2	0	1	4
3	1	1	1

τ_8 :

ID	c_2	c_6	c_7	n Sol
0	0	0	0	16
1	1	0	0	4
2	0	1	0	4
3	1	1	0	1
4	0	0	1	4
5	1	0	1	2
6	0	1	1	0
7	1	1	1	0

τ_9 :

ID	c_1	c_2	c_3	n Sol
0	0	0	0	16
1	1	0	0	4
2	0	1	0	4
3	1	1	0	2
4	0	0	1	4
5	1	0	1	2
6	0	1	1	2
7	1	1	1	1

Figure: Handcrafted #SAT example-run from Markus Zisser[6]

=¿ Dynamic programming Grafiken / Verlaufsschema Kurz erklären -¿ Beweise dazu sind aufwändig und recht speziell /

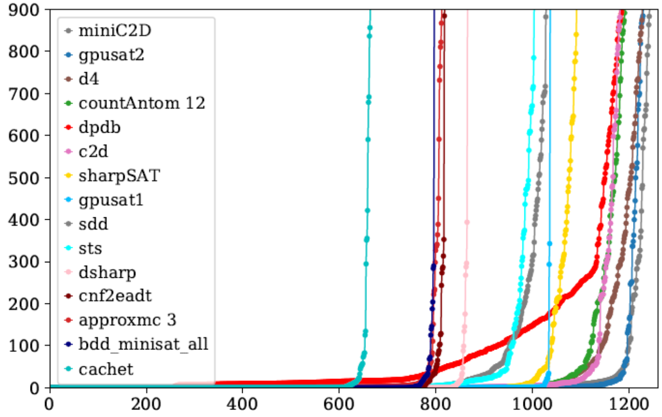
[6]"Solving #SAT on the GPU with Dynamic Programming and OpenCL".

Outlook

for relevant problems the static graph visualization will become to complicated.
https://data-science-blog.com/blog/2015/07/20/3d-visualisierung-von-graphen/
=¿ Automatische Methoden werden häufig schwerer als gedacht.
Für tiefere Debugging Tasks müsste evtl auch der Ansatz erneuert werden
=¿ Was wären weitere Fragestellungen was man ansehen möchte

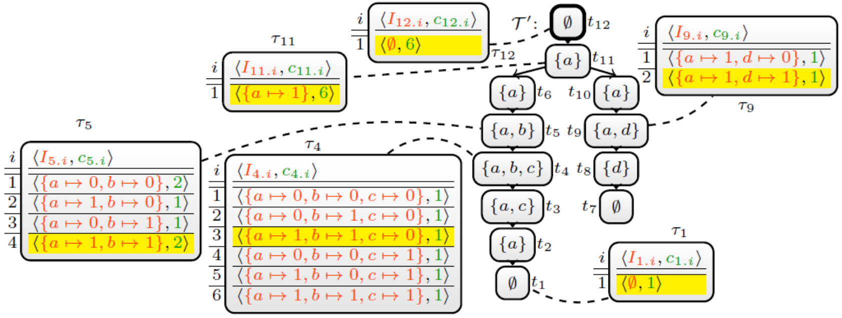
Benchmark

Performance of all three programs on #SAT instances:



Visualization

Manually for dpdb



i	$\langle I_{12.i}, c_{12.i}\rangle$
1	$\langle\emptyset, 6\rangle$

i	$\langle I_{11.i}, c_{11.i}\rangle$
1	$\langle\{a \mapsto 1\}, 6\rangle$

i	$\langle I_{9.i}, c_{9.i}\rangle$
1	$\langle\{a \mapsto 1, d \mapsto 0\}, 1\rangle$
2	$\langle\{a \mapsto 1, d \mapsto 1\}, 1\rangle$

i	$\langle I_{5.i}, c_{5.i}\rangle$
1	$\langle\{a \mapsto 0, b \mapsto 0\}, 2\rangle$
2	$\langle\{a \mapsto 1, b \mapsto 0\}, 1\rangle$
3	$\langle\{a \mapsto 0, b \mapsto 1\}, 1\rangle$
4	$\langle\{a \mapsto 1, b \mapsto 1\}, 2\rangle$

i	$\langle I_{4.i}, c_{4.i}\rangle$
1	$\langle\{a \mapsto 0, b \mapsto 0, c \mapsto 0\}, 1\rangle$
2	$\langle\{a \mapsto 0, b \mapsto 1, c \mapsto 0\}, 1\rangle$
3	$\langle\{a \mapsto 1, b \mapsto 1, c \mapsto 0\}, 1\rangle$
4	$\langle\{a \mapsto 0, b \mapsto 0, c \mapsto 1\}, 1\rangle$
5	$\langle\{a \mapsto 1, b \mapsto 0, c \mapsto 1\}, 1\rangle$
6	$\langle\{a \mapsto 1, b \mapsto 1, c \mapsto 1\}, 1\rangle$

i	$\langle I_{1.i}, c_{1.i}\rangle$
1	$\langle\emptyset, 1\rangle$

Figure: Handcrafted #SAT example-run from dpdb[7]

[7] "Exploiting Database Management Systems and Treewidth for Counting", Fichte, Hecher, Thier, Woltran

Bibliography