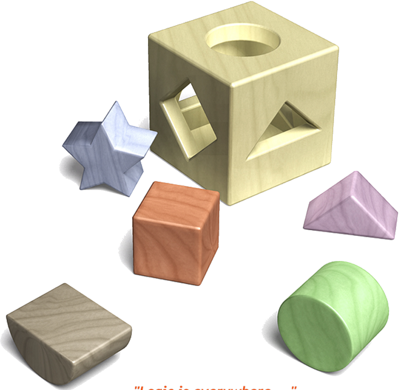
Visualizing Dynamic Programming On Tree Decompositions

Martin Röbke
International Center for Computational Logic
Technische Universität Dresden
Germany

- WHAT **was the motivation**
- WHAT **could be used previously?**
- WHO **benefits from visualization?**
- CHALLENGES **and solutions**
- WHAT **could be developed next?**



"Logic is everywhere ..."

Task definition

Investigate how to automatically visualize dynamic programming algorithms based on existing implementations. Integrate your tool into at least one existing implementation, explain details on your implementation, how the visualization works, and show how this can be used for debugging algorithms.

Martin Röbke
Johannes Fichte
Stefan Gumhold

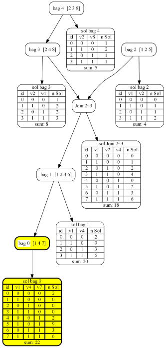
Motivation

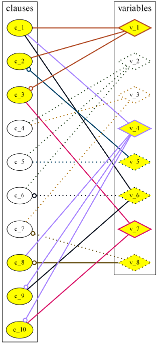
Dynamic programming algorithms can be used to solve combinatorial problems such as SAT, Model Counting, and various graph problems.

Recent research showed that implementations of dynamic programming algorithms can also compete with modern solvers.

Unfortunately, those algorithms are fairly hard to implement as they involve bit fiddling to make them run efficiently.

Creating Visualization for:





Improving

- **examples for students**
- **debugging and improving interaction of complex data-structures**
- **hotspots**

https://gephi.org/
https://tulip.labri.fr/TulipDrupal/
https://neo4j.com/developer/tools-graph-visualization/
Neovis.js Popoto.js Vis.js Sigma.js ...
Commercially licensed: https://www.kineviz.com/graphxr/
Dynamic Data Modeling, Time Series, Discover correlations, trends, and clusters.
https://github.com/vasturiano/3d-force-graph

Existing Visualization

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_4 :

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

JOIN $\longrightarrow$

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

JOIN $\longleftarrow$

IF $\uparrow$

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

$\uparrow$

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

IF $\uparrow$

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

JOIN $\longrightarrow$

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

JOIN $\longleftarrow$

$\uparrow$

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

IF $\uparrow$

$\uparrow$

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

IF $\uparrow$

Figure: Handcrafted #SAT example-run from Markus Zisser[1]

[1] ”Solving #SAT on the GPU with Dynamic Programming and OpenCL”, Diploma Markus Zisser 2018 Technische Universitï¿½t Wien, p.33

Existing Visualization

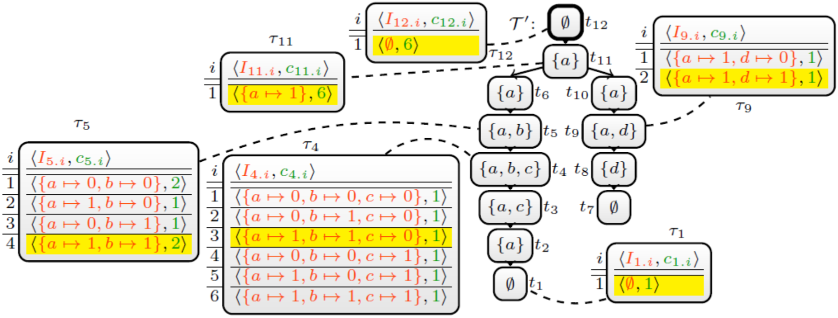


Figure: Handcrafted #SAT example-run from dpdb[2]

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

Background

(Weighted) Model-Counting

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}}

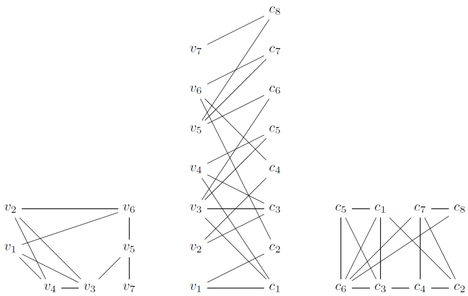


Figure: The primal (left), incidence (middle) and dual (right) graph

Tree Decompositions

Parameterized Complexity and its Applications in Practice
From Foundations to Implementations
Johannes K. Fichte
TU Dresden, Germany
Jakarta, Indonesia
Summer 2019 (May 6th - May 16th) pages 162-174

Backup: VC tree vs graph - example p69, 128

Example: Vertex-Cover problem

Courcelle's theorem

For all $k \in \mathbb{N}$ and MSO-sentences F is the decision problem for a given graph G, whether $G \models F$ is true, in time $2^{p(tw(G))} \cdot |G|$ with a polynom p decidable.

- **drawback: still expensive ($2^{p(twG)}$, $2^{2^{(\#Q)}}$, large constants)**
- **usage:**

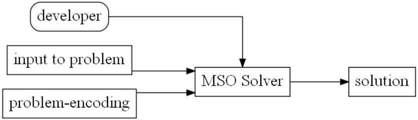


Figure: Implementation of the theorem

gpuSAT2 - Improving Upon Previous Ideas

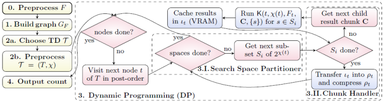
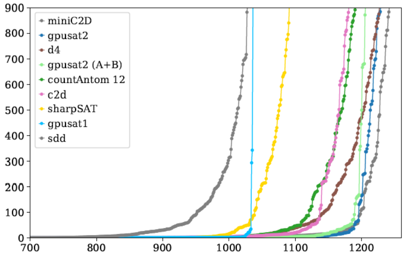


Figure 1: Architecture of our DP-based solver for parallel execution. Yellow colored boxes indicate tasks that are required as initial step for the DP-run or to finally read the model count from the computed results. The parts framed by a dashed box illustrate the DP-part. Boxes colored in red indicate computations that run on the CPU. Boxes colored in blue indicate computations that are executed on the GPU (with waiting CPU).

- ▶ **only primal graph** (simpler solving DP)
- ▶ **customized tree decompositions**
- ▶ **adapted memory-management**
- ▶ **improved precision handling**



dpdb

Using databases for intermediate results

- 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

Challenge2

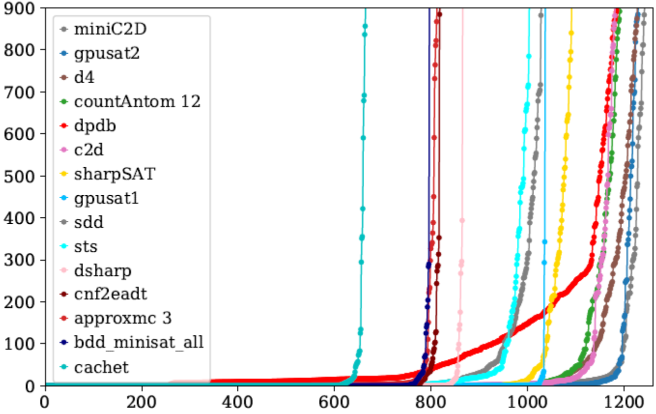
Challenge3

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/

Benchmark

Performance of all three programs on #SAT instances:



Bibliography