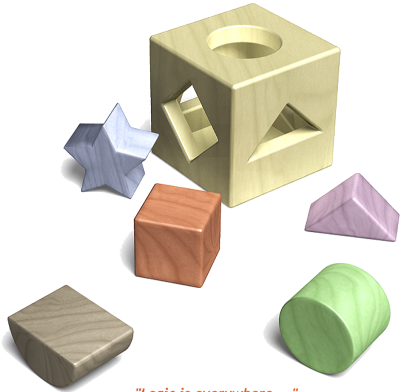
Visualizing Dynamic Programming On Tree Decompositions

Martin Röbke

International Center for Computational Logic
Technische Universität Dresden
Germany

- WHAT **is this about?**
- WHO **benefits from visualization?**



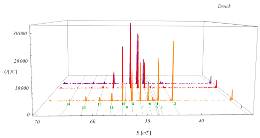
"Logic is everywhere ..."

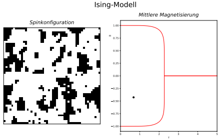
About me

Martin Röbke

- **studying Bachelor CS**
  - **started studying physics at the TU Dresden**
  - **did like logic and visualization more, so switched the faculty** ☺

How did I get to work with my supervisor Johannes Fichte?





Motivation

- **SAT-Problem is NP-complete   #SAT is #P-complete**
  - **Problem with huge instances**
- **Customized algorithms, data-structures, hardware**

Why visualization?

$\longrightarrow$ trace and document the customization

Outlook:

- **Improve and streamline the visualization process**
- **Implement debug-output in existing solvers**
- **Even more dynamic possibilities**

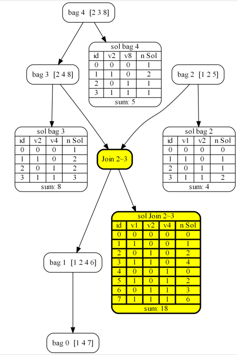


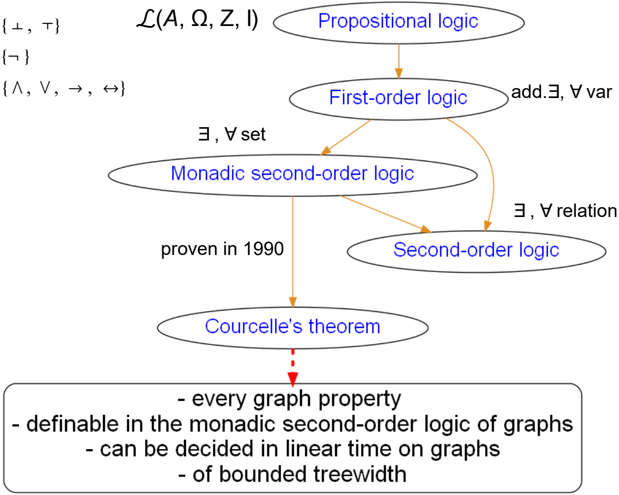
Figure: Example of a #SAT run with DP

Background

$\Omega_0 = \{\perp, \top\}$

$\Omega_1 = \{\neg\}$

$\Omega_2 = \{\wedge, \vee, \rightarrow, \leftrightarrow\}$



Example: Vertex-Cover problem

For the graph **G = (V, E)** we want to compute a set C $\subseteq$ $\boldsymbol{V(G)}$ such that from every edge $\{\boldsymbol{u}, \boldsymbol{v}\}$ there is at least one of $\boldsymbol{u}$ or $\boldsymbol{v}$ in C.

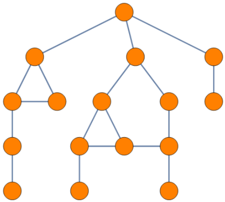


Figure: Example undirected graph **G**

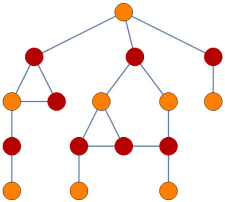


Figure: Minimal VC of **G**

$$\exists \boldsymbol{S} : \forall \{\boldsymbol{u}, \boldsymbol{v}\} \in \boldsymbol{E} : (\boldsymbol{u} \in \boldsymbol{S} \vee \boldsymbol{v} \in \boldsymbol{S})$$

for a given $\boldsymbol{k} \in \mathbb{N}$:
Deciding whether the graph has a vertex cover of size k is NP-complete.

Courcelle's theorem

> *Every graph property definable in monadic second-order logic (MSO) is decidable in linear time on graphs of bounded treewidth.*
>
> *Courcelle, Bruno (1990)*[1]

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

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

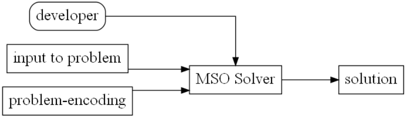


Figure: Implementation of the theorem

[1] Courcelle, Bruno "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs", Information and Computation, 85 (1990) no. 1: 12-75

(Weighted) Model-Counting

The #SAT Problem

Input: boolean formula **F**

Output: the number of satisfying assignments for **F**

The Weighted Model Counting Problem

- $w(lit) \in [0,1], \quad w(\neg lit) = 1 - w(lit)$
- $w(assignment) = \prod_{lit} w(lit)$
- $WMC(formula) = \sum_{satisfying \ \ assignments} w(assignment)$

```
w 665 0.934636
w 666 0.227086
w 667 0.715311
w 668 0.287284
w 669 0.718119
w 670 0.00521478
w 671 0.687858
w 672 0.585033
c clauses added for node v_1_2
-1 2 0
1  -2 0
c clauses added for node v_1_3
-2  -3 0
2 3 0
c clauses added for node v_1_4
-3 -149 4 0
-3 149 -4 0
3 -150 4 0
3 150 -4 0
c clauses added for node v_1_5
-4 -151 5 0
-4 151 -5 0
4 -152 5 0
4 152 -5 0
c clauses added for node v_1_6
-5 -153 6 0
-5 153 -6 0
5 -154 6 0
5 154 -6 0
c clauses added for node v_1_7
-6  -7 0
6 7 0
```

Example for WMC

- **Example set of CNF-clauses:**
  $\{c1 = \{v1, v3, \neg v4\}, c2 = \{\neg v1, v6\}, c3 = \{\neg v2, \neg v3, \neg v4\}, c4 = \{\neg v2, v6\}, c5 = \{\neg v3, \neg v4\}, c6 = \{\neg v3, v5\}, c7 = \{\neg v5, \neg v6\}, c8 = \{v5, v7\}\}$
- **Example weights:**
  $w(v1) = 0.8,\ w(v2) = 0.2,\ w(v3) = 0.1,\ w(v4) = 0.7,\ w(v5) = 0.4,\ w(v6) = 0.5,\ w(v7) = 0.5$
- **Corresponding CNF**
  $(v1 \lor v3 \lor \neg v4) \land (\neg v1 \lor v6) \land (\neg v2 \lor \neg v3 \lor \neg v4) \land (\neg v2 \lor v6) \land (\neg v3 \lor \neg v4) \land (\neg v3 \lor v5) \land (\neg v5 \lor \neg v6) \land (v5 \lor v7)$
- **Satisfying assignments:**

v1	v2	v3	v4	v5	v6	v7
1	1	0	1	0	1	1
1	1	0	0	0	1	1
1	0	0	1	0	1	1
1	0	0	0	0	1	1
0	1	0	0	0	1	1
0	0	1	0	1	0	1
0	0	1	0	1	0	0
0	0	0	0	1	0	1
0	0	0	0	1	0	0
0	0	0	0	0	1	1
0	0	0	0	0	0	1

- **Resulting weighted model count:**
  0.13218

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}}

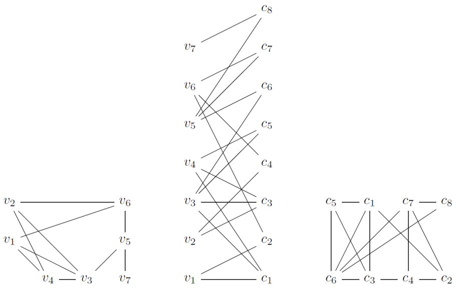


Figure: The primal (left), incidence (middle) and dual (right) graph

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

gpuSAT1

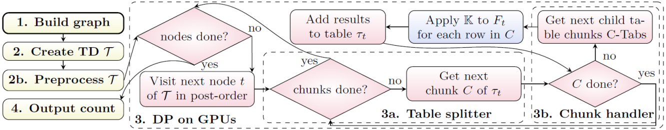


Figure: DP algorithm on the GPU

- **OpenCL**
- **Two operations between bags**

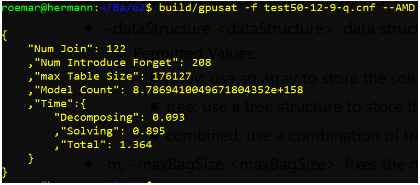


Figure: Example output format from gpuSAT

github: `https://github.com/daajoe/GPUSAT`

gpuSAT2 - Improving Upon Previous Ideas

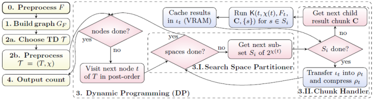
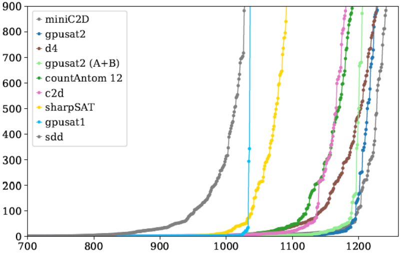


Figure 1: Architecture of our DP-based solver for parallel execution. Yellow colored boxes indicate tasks that are required as initial step for the DP-run or to finally read the model count from the computed results. The parts framed by a dashed box illustrate the DP-part. Boxes colored in red indicate 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
- #o-Coloring
- 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

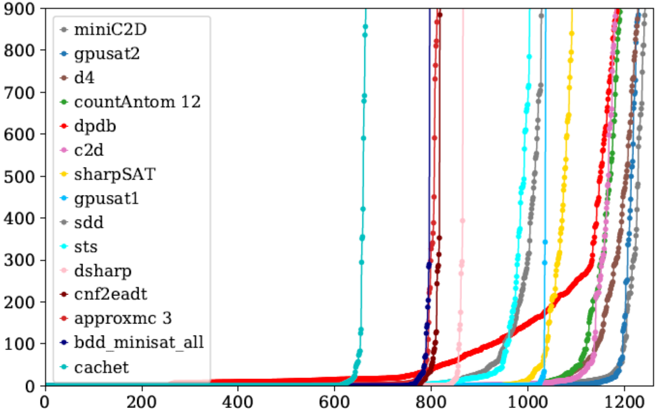
- #εTab#: SELECT 0 AS card
- #intrTab#: SELECT 1 AS val UNION ALL 0
- #localProbFilter#: $([\![u_1]\!]$ OR $[\![v_1]\!])$ AND … AND $([\![u_n]\!]$ OR $[\![v_n]\!])$
- #aggrExp#: MIN(card) AS card
- #extProj#: τ_1.card + … + τ_ℓ.card - $(\Sigma_{i=1}^{\ell}|\chi(t_i) \cap \{a_1\}|$ - 1) * $\tau_1.[\![a_1]\!]$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

(c) Problem MinVC

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

dpdb - Benchmark

Performance of all three programs on #SAT instances:



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 $\longrightarrow$

JOIN $\longleftarrow$

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

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

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 $\longrightarrow$

JOIN $\longleftarrow$

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_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 $\uparrow$

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[2]

[2] "Solving #SAT on the GPU with Dynamic Programming and OpenCL", Diploma Markus Zisser 2018 Technische Universität Wien, p.33

Existing Visualization

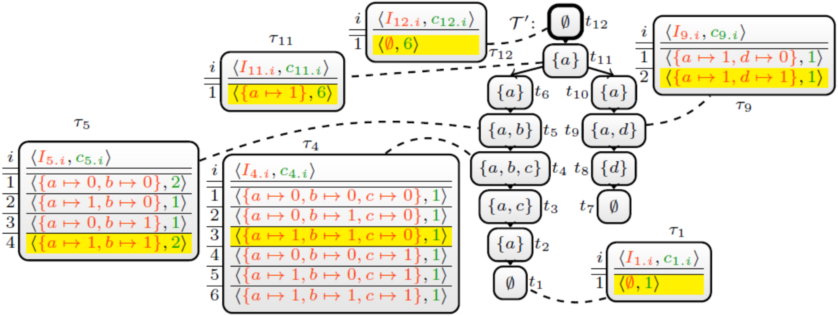


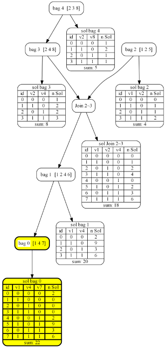
Figure: Handcrafted #SAT example-run from dpdb[3]

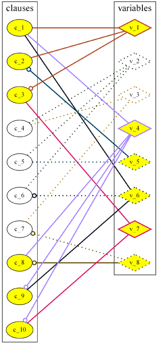
[3] "Exploiting Database Management Systems and Treewidth for Counting", Fichte, Hecher, Thier, Woltran

Creating Visualization for:

Improving

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





Outlook

What can I do?

Visualization

- **customizable output and interactive visualization**
- **possibility of generalizing the underlying graph structure *(Hypergraphs)***
- **reference impl. in CUDA of gpuSAT2**