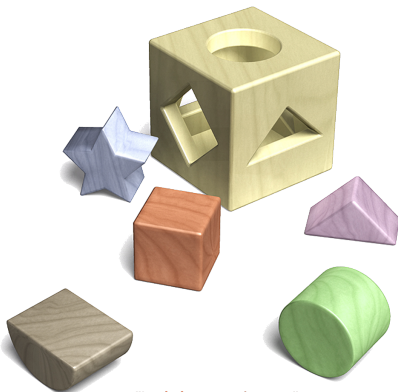


# Visualizing Dynamic Programming On Tree Decompositions

**Martin Rübke**

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 Röbbke  
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.

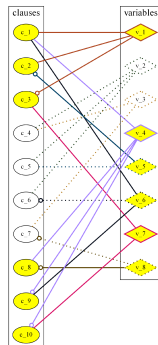
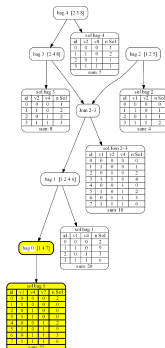
Unfortunately, those algorithms are fairly hard to implement as they involve bit fiddling to make them run efficiently.



# 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>



# Visualization

Manually for gpusat

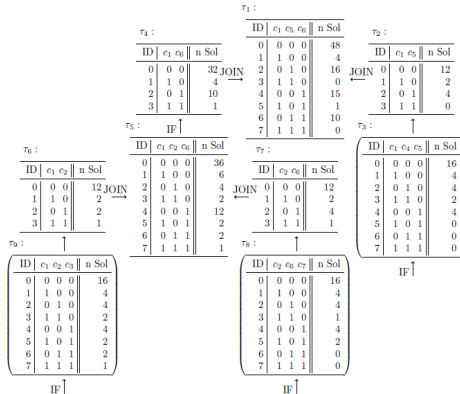


Figure: Handcrafted #SAT example-run from Markus Zisser<sup>1</sup>

<sup>1</sup>"Solving #SAT on the GPU with Dynamic Programming and OpenCL",  
Diploma Markus Zisser 2018 Technische Universität Wien, p.33



# Visualization

Manually for dpdb

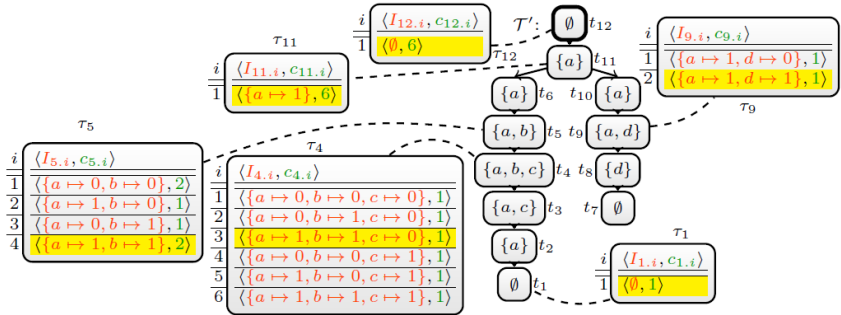


Figure: Handcrafted #SAT example-run from dpdb<sup>2</sup>

<sup>2</sup>"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:

$\{c_1 = \{v_1, v_3, \neg v_4\}, c_2 = \{\neg v_1, v_6\}, c_3 = \{\neg v_2, \neg v_3, \neg v_4\}, c_4 = \{\neg v_2, v_6\}, c_5 = \{\neg v_3, \neg v_4\}, c_6 = \{\neg v_3, v_5\}, c_7 = \{\neg v_5, \neg v_6\}, c_8 = \{v_5, v_7\}\}$

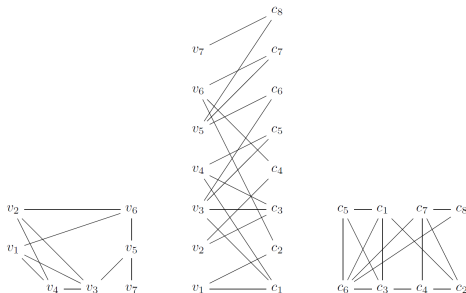


Figure: The primal (left), incidence (middle) and dual (right) graph



# 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 polynomial  $p$  decidable.

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

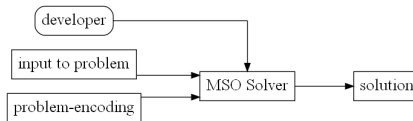


Figure: Implementation of the theorem



# gpuSAT2 - Improving Upon Previous Ideas

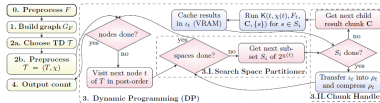
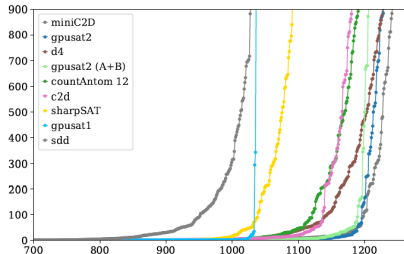


Figure 1: Architecture of our DP-based solver for parallel execution. Yellow colored boxes indicate tasks that are required as initial step for the DP-run or to finally read the model count from the computed results. The parts framed by a dashed box illustrate the DP-part. Boxes colored in red indicate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



## Using databases for intermediate results

```

- #εTab#:          SELECT 1 AS cnt
- #intrTab#:       SELECT 1 AS val UNION ALL 0
- #localProbFilter#: (l1,1 OR ... OR l1,k1) AND ... AND (ln,1 OR ... OR ln,kn)
- #aggrExp#:       SUM(cnt) AS cnt
- #extProj#:       τ1.cnt * ... * τℓ.cnt AS cnt

```

### (a) Problem #SAT

► **SAT**

► **#SAT**

► **Vertex cover**

```

- #εTab#:          SELECT 1 AS cnt
- #intrTab#:       SELECT 1 AS val UNION ALL ... UNION ALL 0
- #localProbFilter#: NOT ([u1] = [v1]) AND ... AND NOT ([un] = [vn])
- #aggrExp#:       SUM(cnt) AS cnt
- #extProj#:       τ1.cnt * ... * τℓ.cnt AS cnt

```

### (b) Problem #o-Col

```

- #εTab#:          SELECT 0 AS card
- #intrTab#:       SELECT 1 AS val UNION ALL 0
- #localProbFilter#: ([u1] OR [v1]) AND ... AND ([un] OR [vn])
- #aggrExp#:       MIN(card) AS card
- #extProj#:       τ1.card + ... + τℓ.card - (Σi=1ℓ |χ(ti) ∩ {a1}| - 1) *
                  τ1. [a1] - ... - (Σi=1ℓ |χ(ti) ∩ {ak}| - 1) * τ1. [ak]

```

### (c) Problem MinVC

github: [https://github.com/hmarkus/dp\\_on\\_dbs](https://github.com/hmarkus/dp_on_dbs)



## Challenge1





## Challenge2



## Challenge3



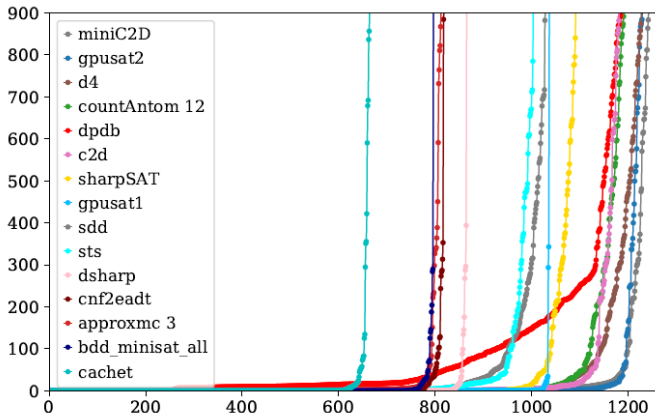
## Outlook

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



