Visualizing Dynamic Programming On Tree Decompositions

Martin Röbke
Fakultät Informatik
Technische Universität Dresden
Germany

- WHAT **is the motivation**
- WHO **benefits from visualization?**
- CHALLENGES **and solutions**
- WHAT **could be used otherwise?**
- OUTLOOK **and ideas**

Motivation

- DP-on-TD-algorithms can solve Model Counting and various combinatorial problems and are provable efficient at it
- Implementations of those are competing with modern solvers
- **But:** those are fairly tedious to implement efficiently
- Practical program information quickly becomes very large (GB)
- Often bugs in the implementation

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].
-¿ Show model counting in a tree example

[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 Decomposition

Gives the DP algorithm a partial ordering for sub-problems.

1. **Each vertex must occur in some bag**

Tree Decomposition

Gives the DP algorithm a partial ordering for sub-problems.

1. **Each vertex must occur in some bag**
2. **For each edge, there is a bag containing both endpoints**

Tree Decomposition

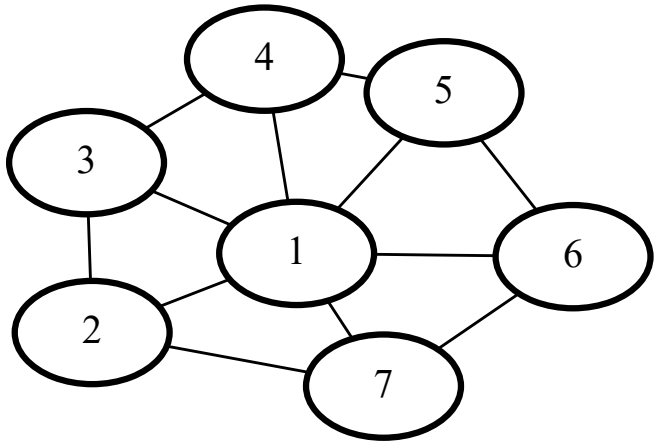
Gives the DP algorithm a partial ordering for sub-problems.

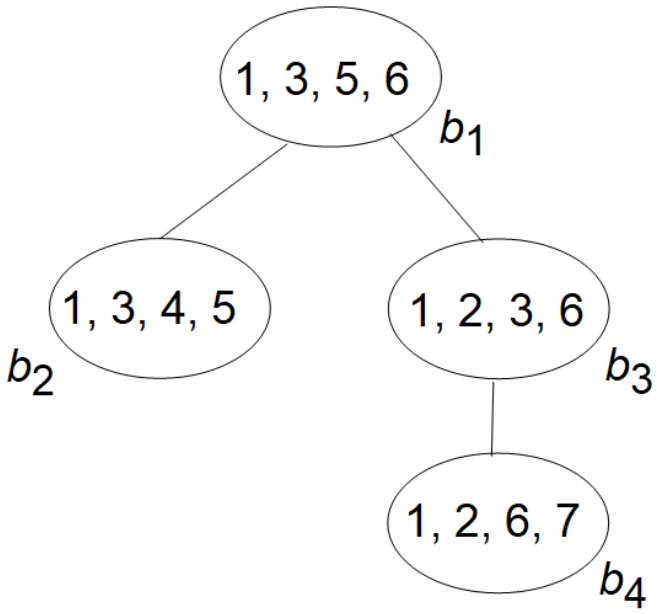
1. **Each vertex must occur in some bag**
2. **For each edge, there is a bag containing both endpoints**
3. **Connected: Subgraph “restricted” to any vertex must be connected**

Tree Decomposition

Gives the DP algorithm a partial ordering for sub-problems.

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**





Width of TD = largest bag -1

Graphs for Boolean Formulas

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

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

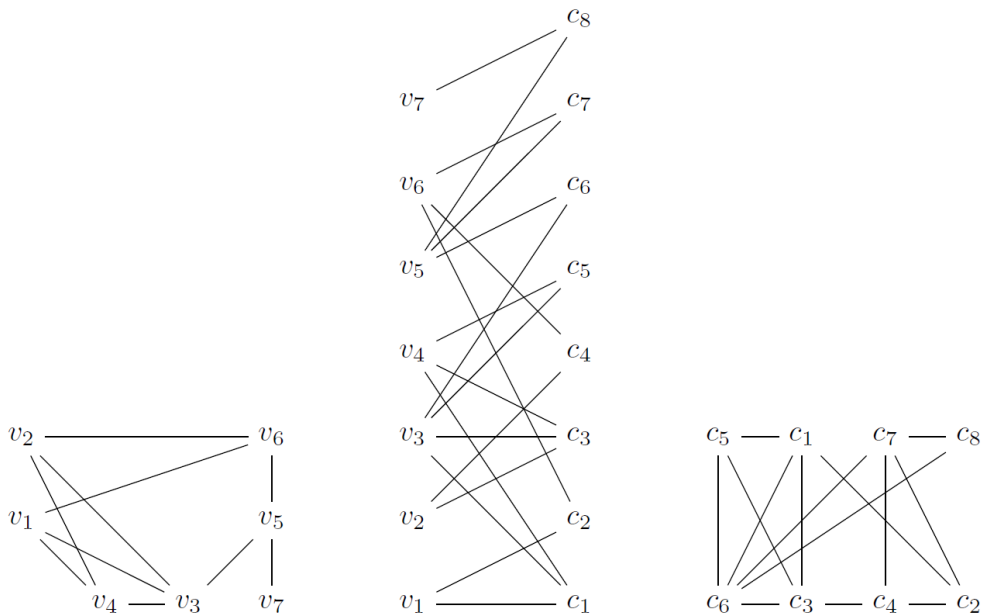
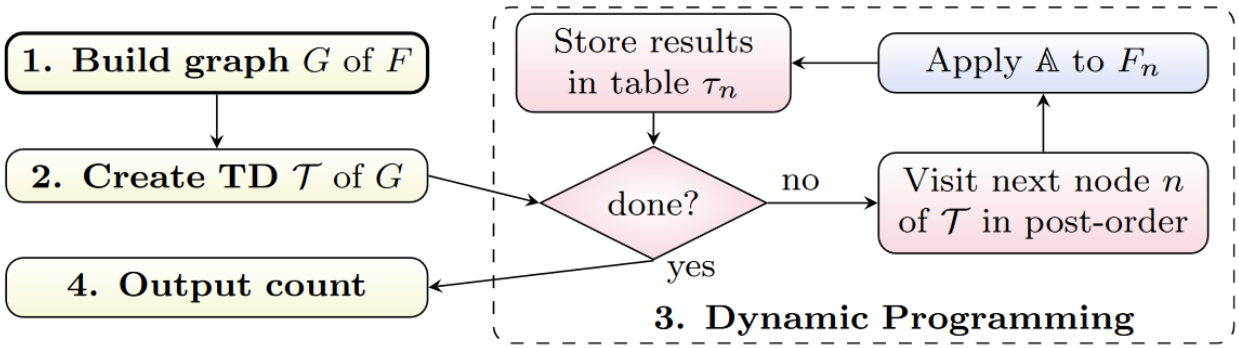
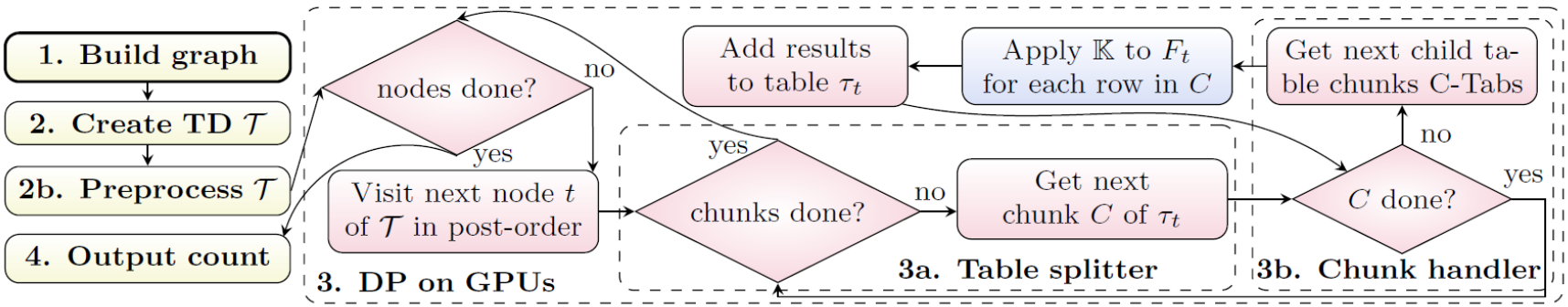


Figure: The primal (left), incidence (middle) and dual (right) graph

gpusat2 - Solving #SAT on GPU



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



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

dpdb

Database templates in Python
Generating SQL queries

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

- #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 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) *` $\tau_1.[\![a_k]\!]$

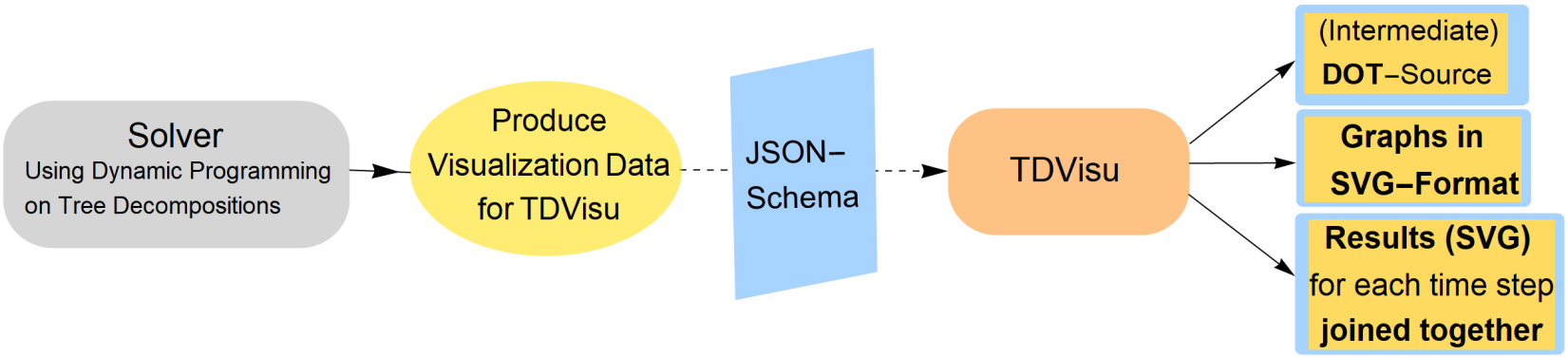
(b) Problem MinVC

- **SAT and #SAT**
- **#o-Coloring**
- **Vertex cover**

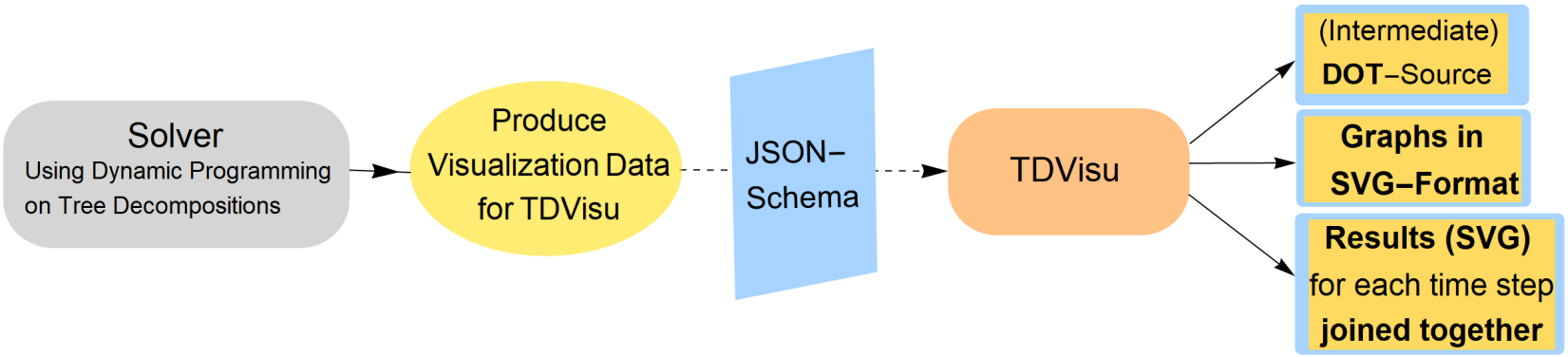
. . .

[1] "Exploiting Database Management Systems and Treewidth for Counting", Johannes Fichte et al. doi: 10.1007/978-3-030-39197-3_10.

Running tdvisu

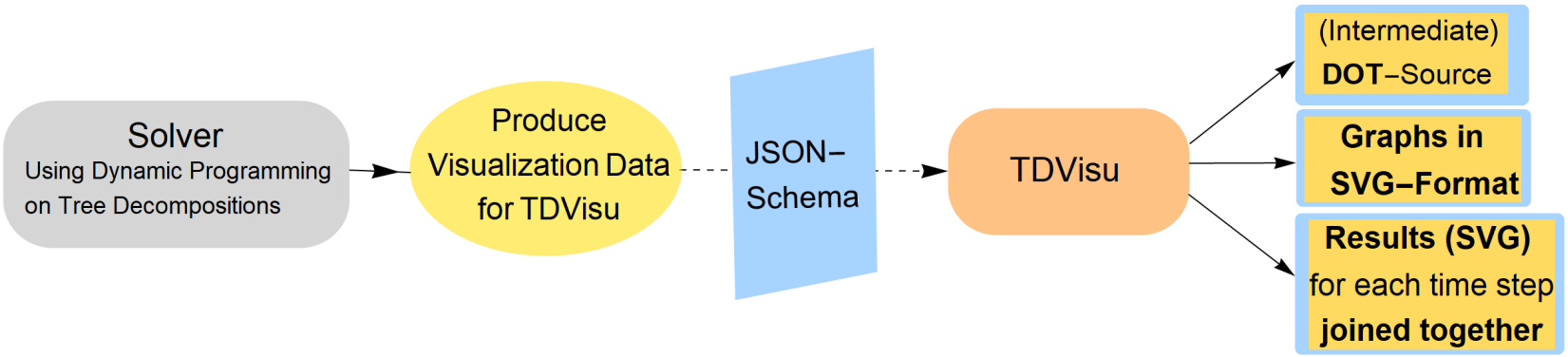


Running tdvisu

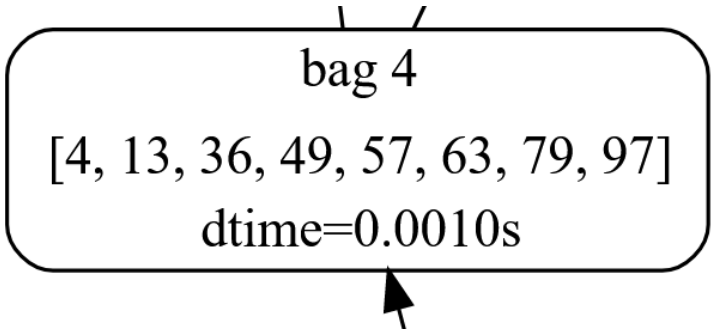


TDVisu producing flexible and further processable formats

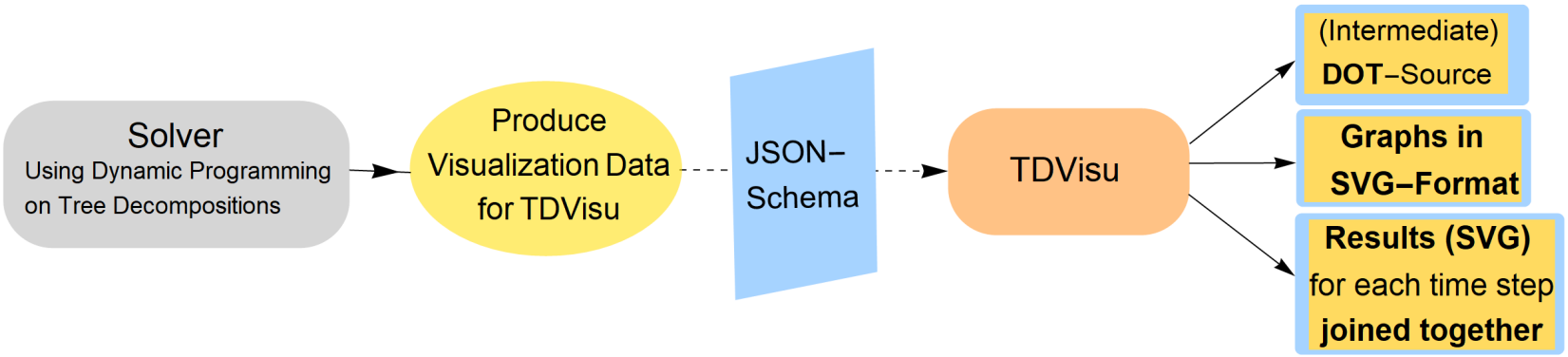
Running tdvisu



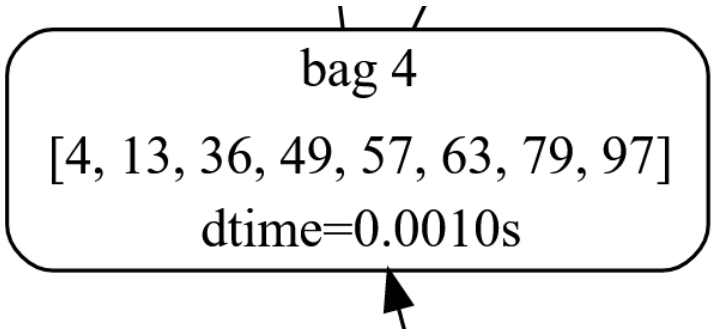
TDVisu producing flexible and further processable formats



Running tdvisu



TDVisu producing flexible and further processable formats



sol bag 4						
v4	v13	v49	v57	v63	v97	size
0	1	0	1	1	1	82
1	0	1	1	1	1	83
1	1	0	1	1	1	82
1	1	0	1	0	1	82
1	1	1	0	1	1	82
1	1	1	1	1	1	83
1	0	0	1	1	1	82
1	1	1	1	1	0	83
1	1	1	1	0	1	83
0	1	1	1	1	1	83
1	1	0	0	1	1	81
1	1	0	1	1	0	82
min-size: 81						

Challenge2

- Wie robust ist die Datenverarbeitung in der Visu Restrictions in strings for ids - Was Gedanken bei der Visu waren

Visualization in Action

MinVC example size 90 (expected 82)

Visualization in Action

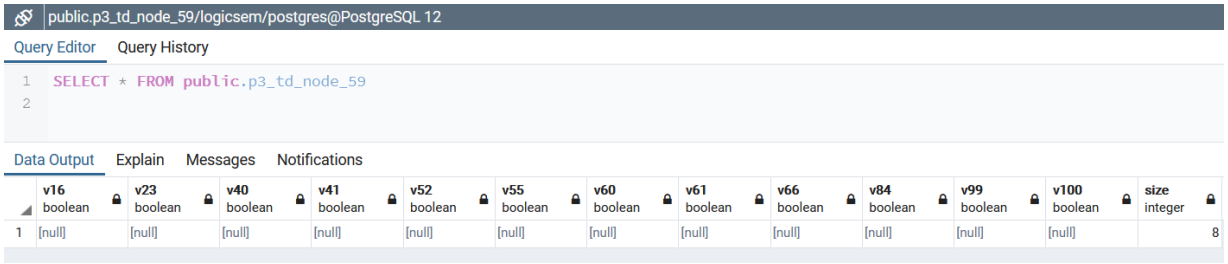
1. **Inspect visualization**

Visualization in Action

1. **Inspect visualization**
2. **Verify findings in solver (in this case dpdb)**

Visualization in Action

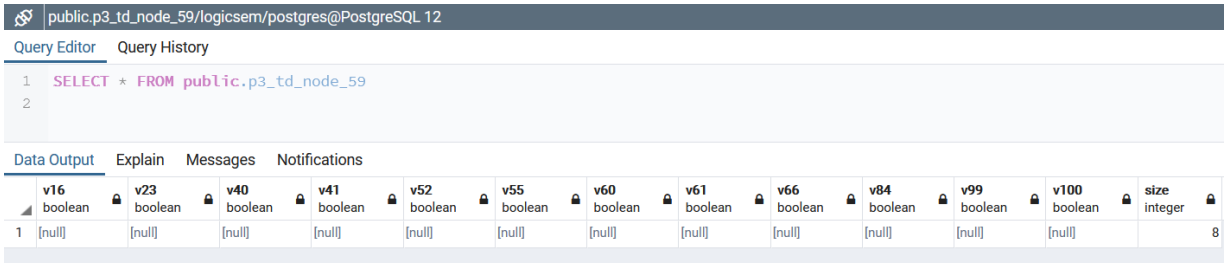
1. **Inspect visualization**
2. **Verify findings in solver (in this case dpdb)**



	v16 boolean	v23 boolean	v40 boolean	v41 boolean	v52 boolean	v55 boolean	v60 boolean	v61 boolean	v66 boolean	v84 boolean	v99 boolean	v100 boolean	size integer
1	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	8

Visualization in Action

1. **Inspect visualization**
2. **Verify findings in solver (in this case dpdb)**



	v16 boolean	v23 boolean	v40 boolean	v41 boolean	v52 boolean	v55 boolean	v60 boolean	v61 boolean	v66 boolean	v84 boolean	v99 boolean	v100 boolean	size integer
1	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	8

3. **Cross reference with standalone tree-decomposition**

Visualization in Action

1. **Inspect visualization**
2. **Verify findings in solver (in this case dpdb)**

public.p3_td_node_59/logicsem/postgres@PostgreSQL 12

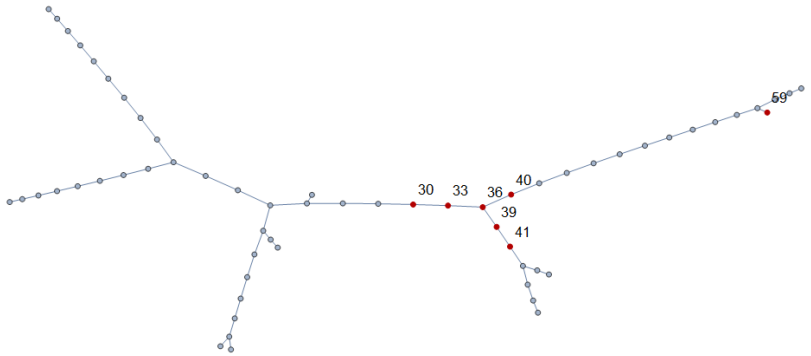
Query Editor | Query History

```
1  SELECT * FROM public.p3_td_node_59
2
```

Data Output | Explain | Messages | Notifications

	v16 boolean	v23 boolean	v40 boolean	v41 boolean	v52 boolean	v55 boolean	v60 boolean	v61 boolean	v66 boolean	v84 boolean	v99 boolean	v100 boolean	size integer
1	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	8

3. **Cross reference with standalone tree-decomposition**



Visualization in Action

1. **Inspect visualization**
2. **Verify findings in solver (in this case dpdb)**

public.p3_td_node_59/logicsem/postgres@PostgreSQL 12

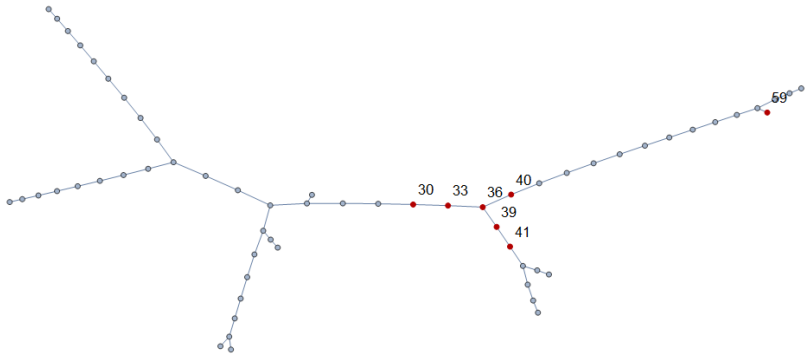
Query Editor | Query History

```
1  SELECT * FROM public.p3_td_node_59
2
```

Data Output | Explain | Messages | Notifications

	v16 boolean	v23 boolean	v40 boolean	v41 boolean	v52 boolean	v55 boolean	v60 boolean	v61 boolean	v66 boolean	v84 boolean	v99 boolean	v100 boolean	size integer
1	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	8

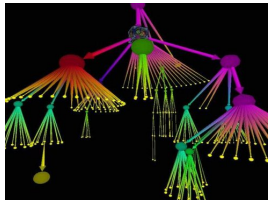
3. **Cross reference with standalone tree-decomposition**



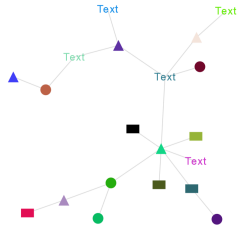
4. **Fix the root cause**

Related Work

- Fichte, Johannes & Hecher, Markus & Morak, Michael & Woltran, Stefan. (2018). Exploiting Treewidth for Projected Model Counting and Its Limits. 10.1007/978-3-319-94144-8_11.
- Hecher, Markus. (2020). Treewidth-aware Reductions of Normal ASP to SAT - Is Normal ASP Harder than SAT after All?. 485-495. 10.24963/kr.2020/49.
- 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



Gephi.org[2] Tulip [3]



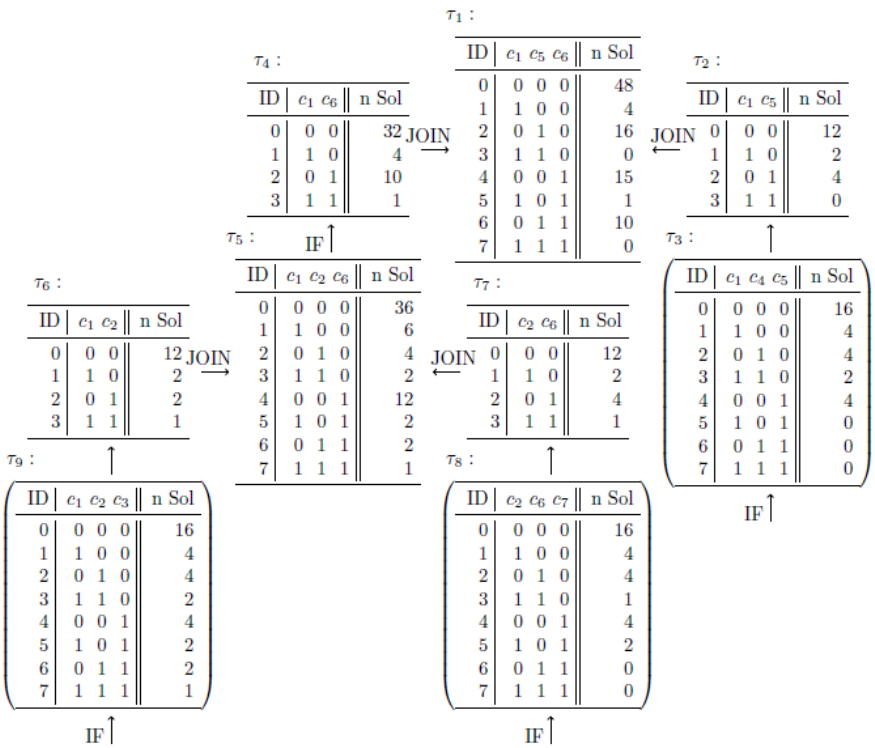
[4] Vis.js Sigma.js vasturiano/3d-force-graph [5]

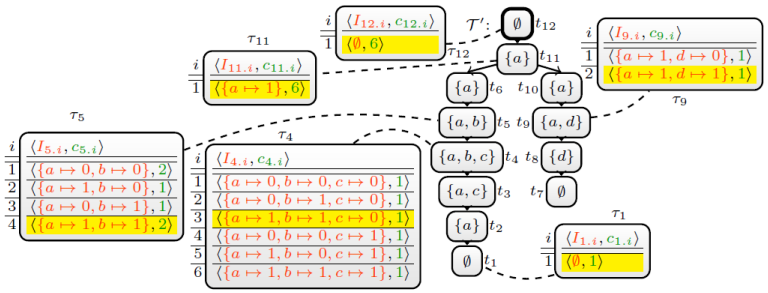
Briefly looked up different formats and graph software. With the diverse / large node labels and special layout the creation of a lightweight and customizable exchange format took precedence over the integration into special layout software.

[2] https://gephi.org/ - Tool for data analysts and scientists keen to explore and understand graphs.
[3] tulip.labri.fr/TulipDrupal/ - Better Visualization Through Research.
[4] https://neo4j.com/developer/tools-graph-visualization/
[5] https://github.com/vasturiano/3d-force-graph

Visualization

Manually for one run





[5] "Exploiting Database Management Systems and Treewidth for Counting", Johannes Fichte et al. doi: 10.1007/978-3-030-39197-3_10.

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

Outlook

Static $\rightarrow$ Dynamic

Interesting Questions :

- **Utilizing graph databases for visualization and queries for debugging**
- **Enrich the visualization with debugging info for each node**
- **Cross reference the creation of rows in parent nodes**
- **For more advanced debugging tasks you may also need to revise the approach**

Summary

This thesis created tdvisu as a tool that

- **integrates into existing implementations**
- **statically exports data from runs**
- **compiles simple DOT files and SVG graphics**

For further research it provides

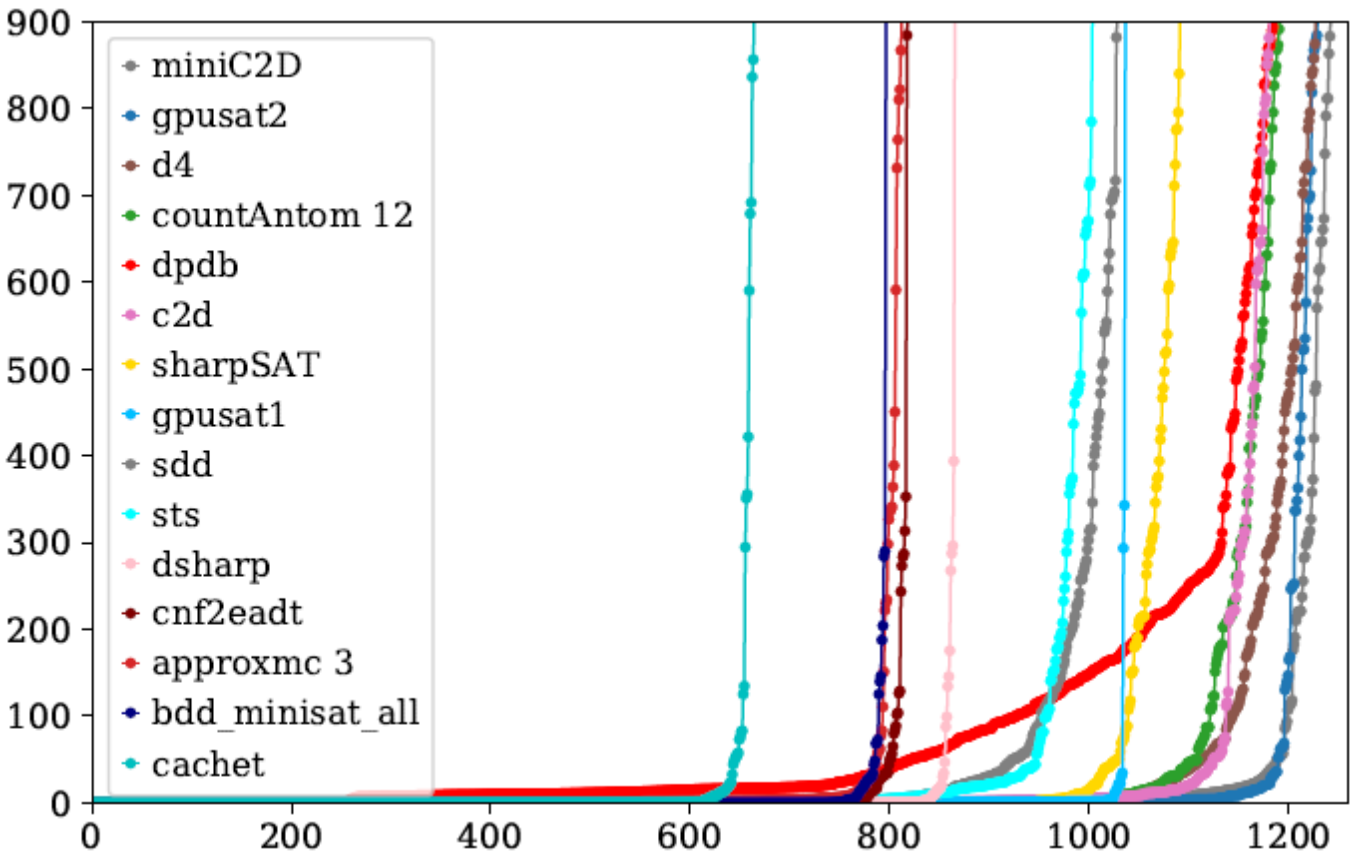
- **starting point for more complex investigations of**
  - **bug spotting**
  - **and fixing by using visualizations**

Bibliography

See the citations in the thesis.

Benchmark

Performance of all three programs on #SAT instances:



Visualization

Manually for dpdb

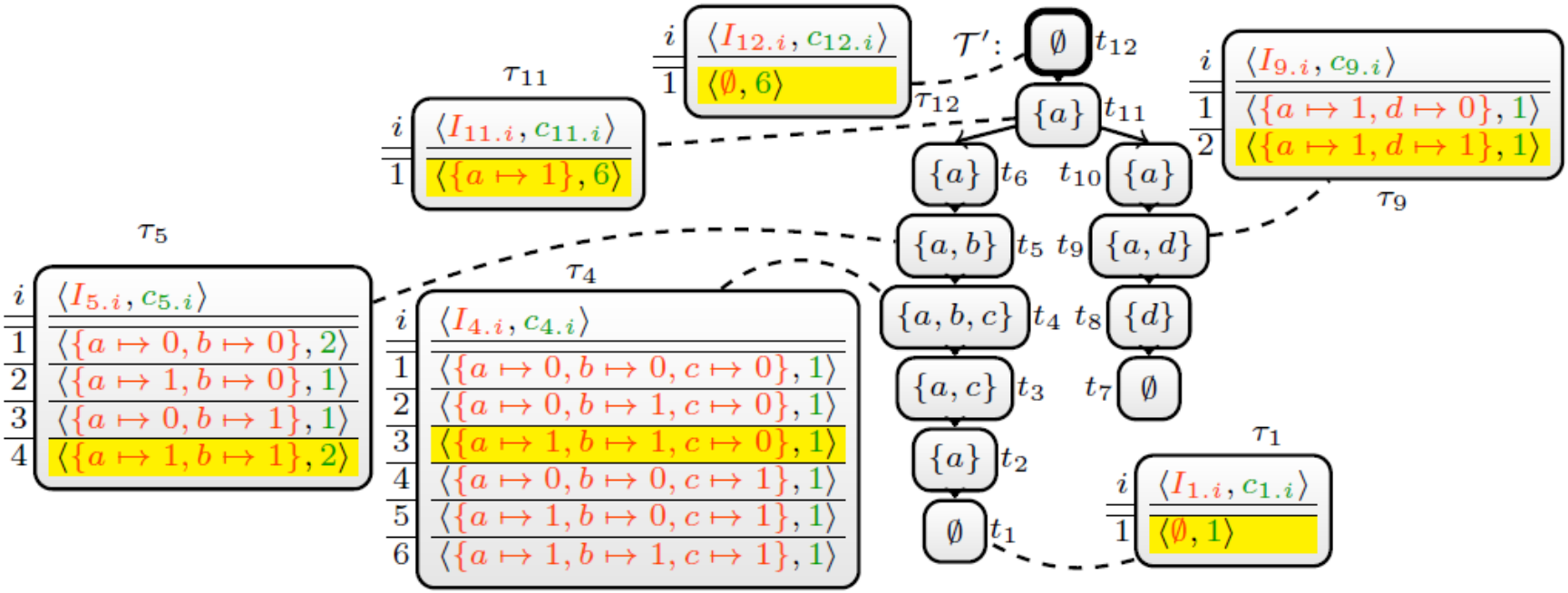


Figure: Handcrafted #SAT example-run from dpdb[6]

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