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**
- **Model counting is extremely space intensive (much more than SAT)**
- **Often bugs in the implementation**

Contribution

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**

Background

The algorithms of interest solve problems of:

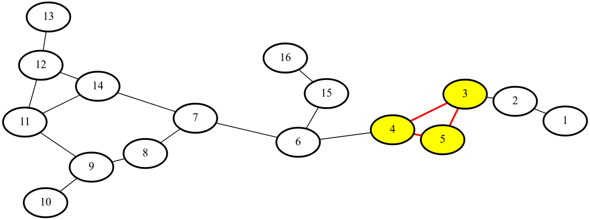
- **combinatorics (NP-problems)**
- **model-counting (#P-problems) -** Instead of **one** solution we want to count **all** solutions

Background

The algorithms of interest solve problems of:

- **combinatorics (NP-problems)**
- **model-counting (#P-problems) -** Instead of **one** solution we want to count **all** solutions

Example of two snapshots of getting a minimal vertex cover via DP:

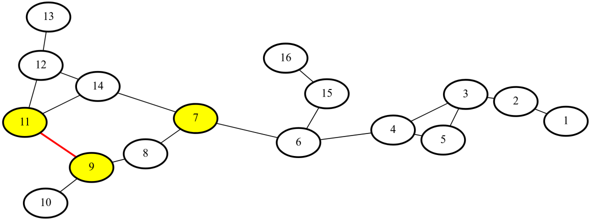


Background

The algorithms of interest solve problems of:

- **combinatorics (NP-problems)**
- **model-counting (#P-problems) -** Instead of **one** solution we want to count **all** solutions

Example of two snapshots of getting a minimal vertex cover via DP:



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

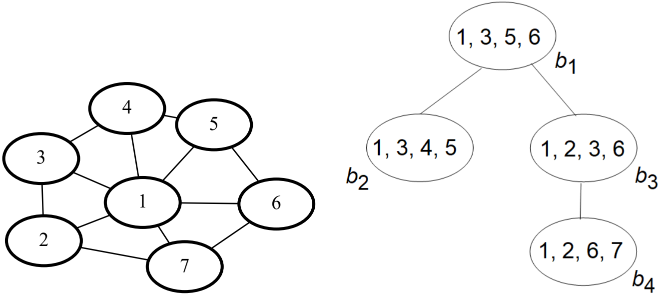
Gives the DP algorithm a partial ordering for sub-problems.

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



Width of a TD is: size of largest bag -1

width $=$ **3**

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}}

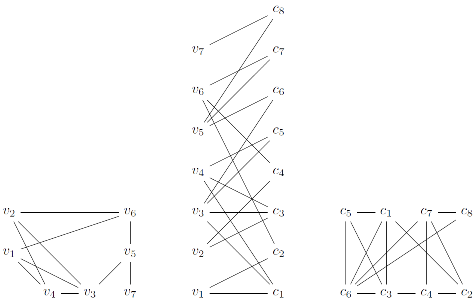
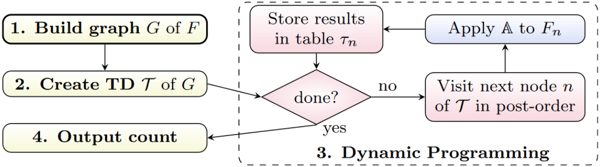


Figure: The primal (left), incidence (middle) and dual (right) graph

gpusat2 - Solving #SAT on GPU



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

---

[0] 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` $[\![v_n]\!])$
- #extProj#: τ_1`.card + ... +` τ_ℓ`.card - (`$\Sigma_{i=1}^{\ell}|\chi(t_i) \cap \{a_1\}|$ `- 1) *` τ_1`.`$[\![a_1]\!]$ `- ... - (`$\Sigma_{i=1}^{\ell}|\chi(t_i) \cap \{a_k\}|$ `- 1) *` τ_1`.`$[\![a_k]\!]$
- #aggrExp#: `MIN(card) AS card`

(b) Problem MinVC

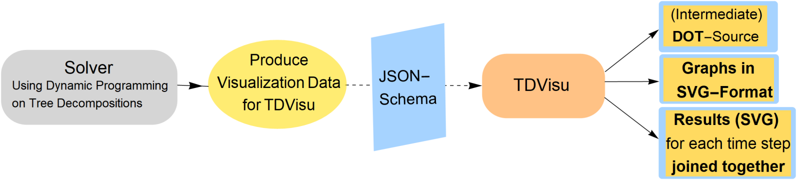
- **SAT and #SAT**
- **#o-Coloring**
- **Vertex cover**

. . .

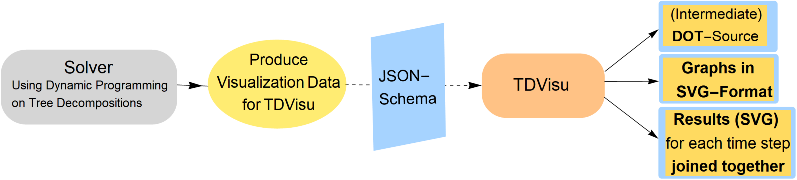
---

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

Running tdvisu

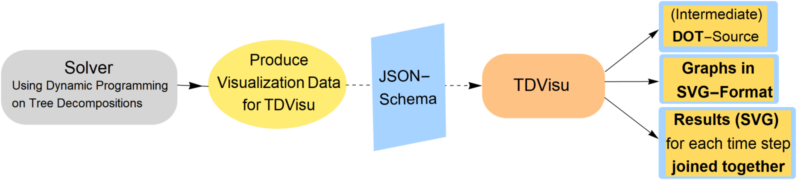


Running tdvisu

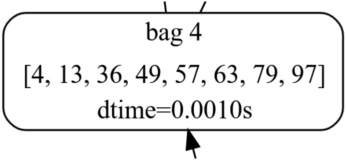


TDVisu producing flexible and further processable formats

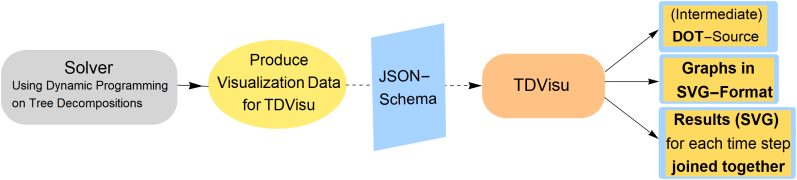
Running tdvisu



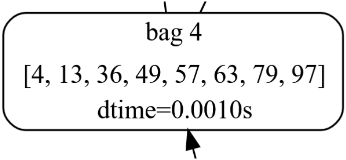
TDVisu producing flexible and further processable formats

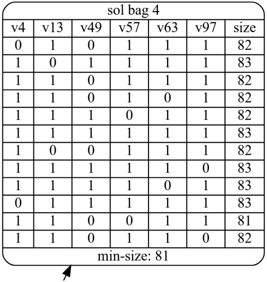


Running tdvisu



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

Capabilities and Limitations

Integration of solvers via TDVisu.schema.json [1]

Capabilities:

- **Extracting basic (extendable) information (TD, solution nodes, order+time of processing...) from gpusat + dpdb**
- **Constructing and enriching the with solver information**
- **Adding multiple graphs for e.g. problems on Boolean formulas**
- **Providing a discrete timeline**
- **Parameters to control the layout and coloring of the data**

Limitations:

- **Can not further animate for example the origin of solutions**
- **Maneuvering in very large graphs is not very ergonomic with static content**

---

[1] raw.githubusercontent.com/VaeterchenFrost/tdvisu/master/TDVisu.schema.json

Visualization in Action

MinVC example size 90 (expected 82)

Visualization in Action

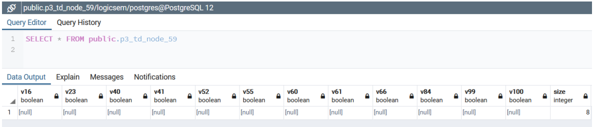
1. **Inspect visualization**

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)**

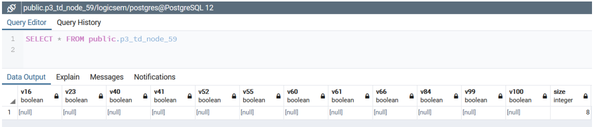


```
SELECT * FROM public.p3_td_node_59
```

	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)**



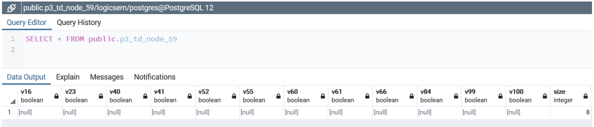
```
SELECT * FROM public.p3_td_node_59
```

	v16	v23	v40	v41	v52	v55	v60	v61	v66	v84	v99	v100	size
	boolean	boolean	boolean	boolean	boolean	boolean	boolean	boolean	boolean	boolean	boolean	boolean	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)**

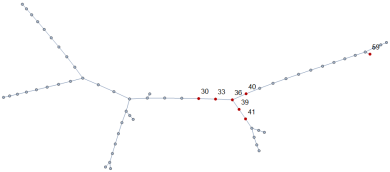


```
SELECT * FROM public.p3_td_node_59

```

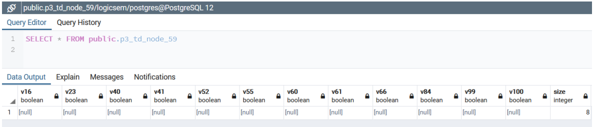
	v16 boolean	v23 boolean	v40 boolean	v41 boolean	v52 boolean	v55 boolean	v60 boolean	v61 boolean	v66 boolean	v84 boolean	v99 boolean	v100 boolean	size integer
1	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	[null]	8

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



Visualization in Action

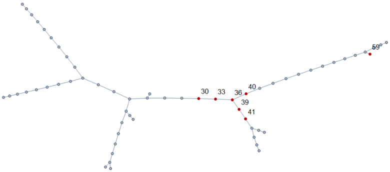
1. **Inspect visualization**
2. **Verify findings in solver (in this case dpdb)**



```sql
SELECT * FROM public.p3_td_node_59
```

	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**



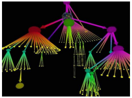
4. **Fix the root cause**

Related Work on the Algorithms

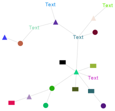
- Fichte, Johannes & Hecher, Markus & Morak, Michael & Woltran, Stefan. (2018). Exploiting Treewidth for Projected Model Counting and Its Limits. 10.1007/978-3-319-94144-8_11.
- Hecher, Markus. (2020). Treewidth-aware Reductions of Normal ASP to SAT - Is Normal ASP Harder than SAT after All?. 485-495.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

Related Work on Visualizations

- **Marie-Christin Harre, Jan Jelschen, and Andreas Winter. "ELVIZ: A querybased approach to model visualization". In: Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI) (Jan. 2014), pp. 105–120.**
- **Stephan Diehl. Software Visualization. Visualizing the Structure, Behaviour, and Evolution of Software. English. Springer, 2007. 199 pp. isbn: 978-3540465041.**
- **Jason Daida et al. "Visualizing Tree Structures in Genetic Programming". In: Genetic Programming and Evolvable Machines 6 (Mar. 2005). doi: 10.1007/s10710-005-7621-2.**



Gephi.org[2] Tulip [3]



[4] Vis.js

Sigma.js

vasturiano/3d-force-graph [5]

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

Outlook

Static → Dynamic

Interesting Questions :

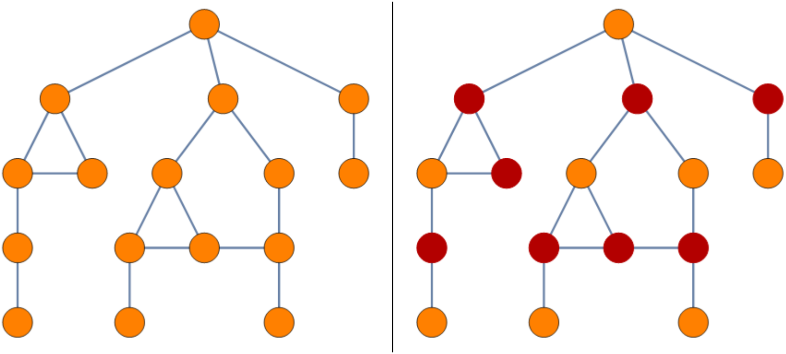
- **Cross reference the creation of rows in parent nodes**
- **Enriching the visualization with more data for each node**
- **For more advanced debugging tasks you may also need to revise the approach**
- **Utilizing graph databases for debugging**

Final slide

Bibliography

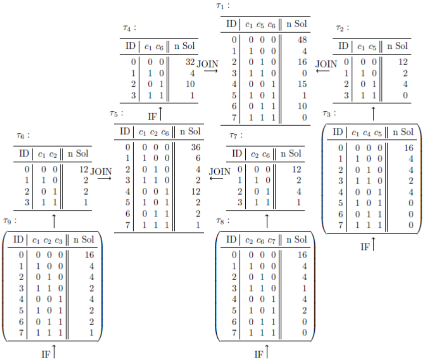
See the citations in the thesis.

MinVC for example graph



Visualization

Manually for one run



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

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_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

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 :

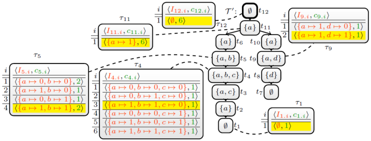
ID	c_2	c_6	n Sol
0	0	0	12
1	1	0	2
2	0	1	4
3	1	1	1

τ_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

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_{12}

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

τ_{11}

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

τ_9

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

τ_5

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	$\langle \{a \mapsto 1, b \mapsto 1\}, 2\rangle$

τ_4

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	$\langle \{a \mapsto 1, b \mapsto 1, c \mapsto 1\}, 1\rangle$

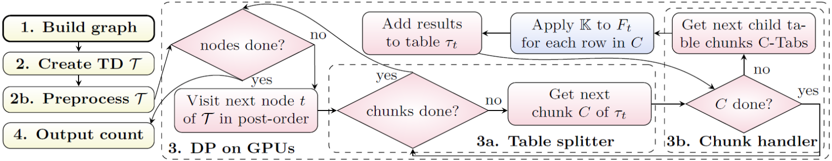
τ_1

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

---

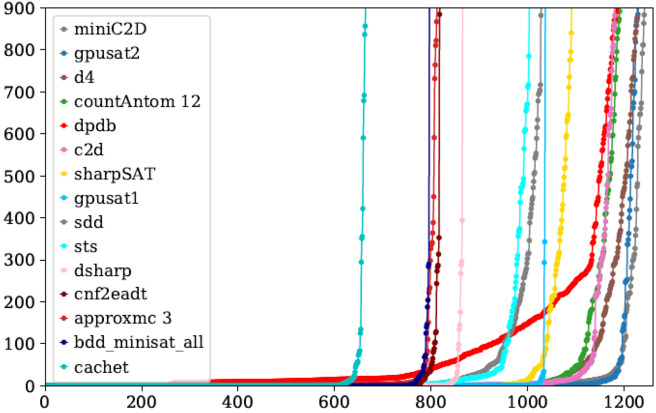
[5] “Exploiting Database Management Systems and Treewidth for Counting”, Johannes Fichte et al. doi: 10.1007/978-3-030-39197-3_10.

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


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

Benchmark

Performance of all three programs on #SAT instances:



ELVIZ - Query based approach to software visualization

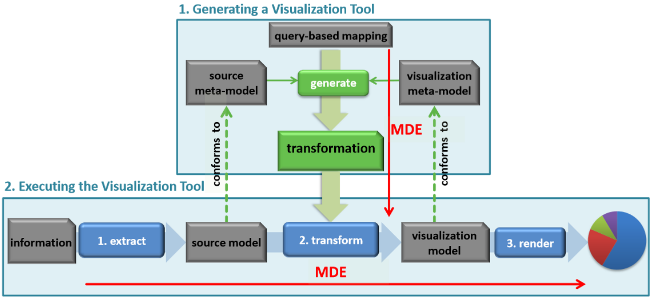


Figure: Overview of the ELVIZ-approach[6]

---

[6] Fig 1 in: Marie-Christin Harre, J. Jelschen, A. Winter. “ELVIZ: A querybased approach to model visualization”. In: Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI) (Jan. 2014), pp. 105–120.