Bachelor Thesis

Visualizing Dynamic Programming on Tree Decompositions

Martin Röbke

born: 04.03.1995 in Dresden, Germany

matriculation number: 3949819

martin.roebke@tu-dresden.de

Technische Universität Dresden
Faculty of Computer Science
International Center For Computational Logic

Supervisor: Dr. Johannes Fichte

Second evaluator: Prof. Dr. rer. nat. Stefan Gumhold

Dresden, June 11, 2020

Abstract

This thesis introduces a practical and lightweight implementation for visualizing dynamic programming on tree decompositions. The provided source code is located in the python-package tdvisu for the purpose of visualizing, teaching and analyzing the dynamic solving process.
It defines a JSON-format specification for portability and customization of the visualization combined in one human-readable file and two reference implementations in actual solvers. The implementation currently does not support hyper-graphs and assumes that each node in the tree decomposition has either one or two children. The visualization output consists by default of scalable vector graphics SVG, a very flexible text-based that can be compressed and modified very easily without loss of quality. The images are split up into different views on the current state of the tree decomposition for consecutive user-defined time steps showing the progress of the dynamic programming. As illustrations of the possibilities for an application smaller examples from the problem-types "#SAT" and "minimal vertex cover" are presented, as well as an example of a faulty tree-decomposition that occurred during development. Intended audience:

- Developer of dynamic programming on tree decompositions for debugging.
- Researcher of such algorithms for comparisons and visualizations.
- Teachers or students looking for automatic visualization of their examples and the dynamic programming.

As two reference implementations of dynamic programming on tree decompositions we chose the projects GPUSAT and dpdb.

Contents

1. Introduction

Graphs are increasingly interesting in scientific work, as the applications of interconnected datasets grow. The use cases outlined by Neo4j [1], a provider of open source database tools for "labeled property graphs" and include fields like

- Network and Database Infrastructure
- Recommendation Engines
- Artificial Intelligence and Analytics

While many problems can be expressed using the language of graphs, there is still plenty of room to utilize graph The idea for this project comes from my supervisor Dr. Johannes Fichte, who works on and with many projects such as the ones listed above on solving monadic second order logic (MSOL [2]) problems using highly parallelized architectures like graphics processing units or state of the art databases. One early implementation is published in [3] where for different real world examples the results looked promising These projects are very competitive ????REF????? for solving even large instances of those problems. The source code for TDVisu is available under GPL3 license.

Graphviz is a open source graph visualization software that provides customizable visualization for directed and undirected graphs. The information processed by graphviz

intro. mit motivation und related work, state of the art, advancements.

Visualization Pipeline

Stand Umsetzung, Tools: Slack, Trello, GitHub, Presentations

2. Background

This chapter provides the reader with a brief background for this work.

We begin with a description on SAT and #SAT as examples for a very general problem that can be described with monadic second order logic (MSOL). Furthermore the general case of MSOL will be described, as well as the *DIMACS*-file-format used in the projects. The following section describes Tree Decompositions (TDs) which are the basis for our visualization. Finally we shortly discuss Courcelle's Theorem [2] as a related method of solving these problems.

2.1. Boolean satisfiability problem

`https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`

SAT was the first known NP-complete problem, shown by Stephen Cook at the University of Toronto in 1971 [4]

literal ≡ boolean variable v or its negation
clause ≡ finite set of literals, interpreted as the disjunction
unit ≡ clause with |c|=1
CNF formula ≡ set of clauses, interpreted as their conjunction
var ≡ set of variables contained in the clause or clause set C
assignment ≡ α:var(C) $\rightarrow$ {0,1}
satisfiedclause ≡ if $\exists$v $\in$ var(c), v$\in$c and α(v)=1 or $\neg$v$\in$c and α(v)= 0. Otherwise falsified
satisfiedform ≡ each clause in the formula is satisfied by assignment

Connection to graphs with [5]

SAT Handbook: Even finding a single solution can be a challenge for such problems; counting the number of solutions is much harder.Not surprisingly, the largest formulas we can solve for the model counting problem with state - of - the - art model counters are orders of magnitude smaller than the formulas we can solve with the best SAT solvers. Generally speaking, current exact counting methods can tackle problems with a couple of hundred variables, while approximate counting methods push this to around 1, 000 variables.

2.2. Monadic Second Order Logic

See also figure 2.

Explain graphs?Node, Edge

MSO graph properties are "fixed-parameter-tractable" with respect to clique-width and tree-width. `https://www.youtube.com/watch?v=hZI-wANHO1w` 5th workshop on Graph Classes, Optimization, and Width Parameters (GROW 2011) 2011-10-28. MSO

counting (k-colorings)and optimizing (distance between two vertices...) functions. Interested in MSO logic over graphs.

Two types of MSO formulas *or logical graph representations.*

- MSO formulas
- MSO_2 formulas with edge quantification ≡ MSO formulas over incidence graphs
- G=(vertices, edges as binary relation)
- INC(G) = (vertices and edges, Inc) for G undirected: Inc(e,v) ¡-¿ v is a vertex of edge e
- FPT for clique width
- FPT for tree-width

This can also be done for directed graphs!

Typical MSO_2 graph properties:

has a perfect matching has a Hamiltonian circuit spanning tree of degree ≤ 3

The expressions have the form: "There exists a **set of edges** that is..." can not be transferred into "set of vertices"

`https://youtu.be/Wyn3djrYg7c?t=1385` Bruno Courcelle: Recognizable sets of graphs: algebraic and logical aspects `https://library.cirm-math.fr/Record.htm?idlist=2&record=19276851124910940339` Recording during the thematic meeting: "Frontiers of reconnaissability" the April 29, 2014 at the Centre International de Rencontres Mathématiques (Marseille, France)

FPT for model checking: An algorithm is FPT if it takes time $f(k) \cdot n^c$ for some fixed function f and constant c. The size of the input is n. The value k is a parameter of the input. This algorithm is then usable for small values of k. usually tree-width and clique width.

2.3. DIMACS Format

Developed in 1993 at Rutgers University. DIMACS (the Center for Discrete Mathematics and Theoretical Computer Science)

Wolfram Language fully supports the DIMACS format for storing a single undirected graph. `https://reference.wolfram.com/language/ref/format/DIMACS.html`

DIMACS CNF: This format is used to define a Boolean expression, written in conjunctive normal form. `https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html`

Other formats for WMC, different graphs...

Supported also in Maple `https://www.maplesoft.com/support/help/maple/view.aspx?path=Formats/CNF`

Examples in appendix

2.4. Tree Decomposition

[5]chapter 2.2 Tree decompositions were originally introduced by Robertson and Seymour [6] in 1984. A *tree decomposition* (TD) of a graph G is a pair (T, χ). T is a tree and χ is a mapping which assigns each node $n \in V(T)$ a set $\chi(n) \subseteq V(G)$ called a *bag*. Then (T, χ). T is a TD if the following conditions hold:

1. for each vertex $v(n) \in V(G)$ there is a node $n \in V(T)$ such that $v \in \chi(n)$
2. for each edge $(x, y) \in E(G)$ there is a node $n \in V(T)$ such that $x, y \in \chi(n)$
3. if $x, y, z \in V(T)$ and y lies on the path from x to z then $\chi(x) \cap \chi(z) \subseteq \chi(y)$.

The width $width(T)$ of a tree decomposition T is $max_{n \in V(T)}(|\chi(n)|) - 1$. The tree width of a graph is the *minimal width* over all tree decompositions of the graph.

!!!Example (can take one from the visualizations) also in [7] page 169

2.5. Courcelle's Theorem

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

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)

The workflow then looks like we see in figure 1.

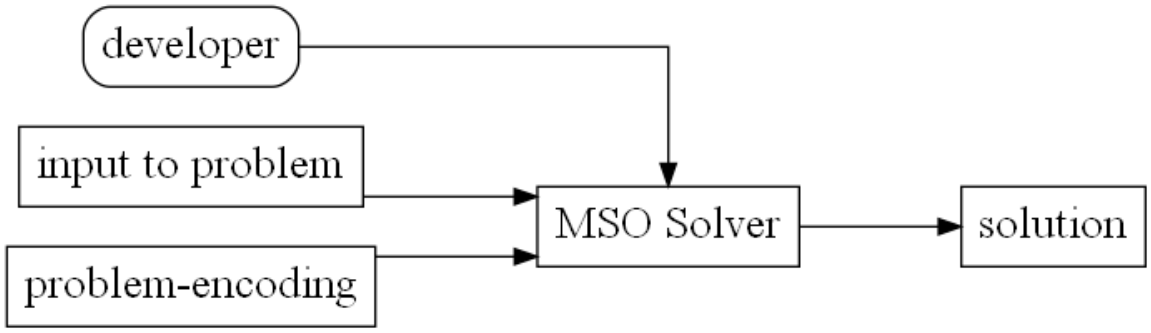

Figure 1: Implementation of the theorem

[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

3. Concept

What I do and why I did it Steps in Trello, Issues, Commits Research: language (python - explain) graph-construction (graphviz vs networkX), examples (diploma at first).

4. My Visualization Project

Pip, requirements files, virtual environments and how to pick quality Python libraries: https://realpython.com/products/managing-python-dependencies/?utm_source=drip&utm_medium=email&utm_campaign=productfooter&utm_content=mpd

The first stages were in https://github.com/VaeterchenFrost/gpusat-VISU and the first releases of the source code outsourced tohttps://github.com/VaeterchenFrost/tdvisu

The objective of this project was/is to support the visualization mainly to document and improve the development efforts of dynamic programming on tree decompositions.

The tree decompositions in every tested application were provided by the utility https://github.com/mabseher/htd (small but efficient C++ library for computing (customized) tree and hypertree decompositions). Files / Classes / Methods Current perspective Checking with www.deepcode.ai

4.1. Integration in GPUSAT

Windows opencl driver problem: ERROR: clBuildProgram(-9999)

Programm `https://github.com/VaeterchenFrost/GPUSAT`
Differences: `https://github.com/daajoe/GPUSAT/compare/master...VaeterchenFrost:master` Commits 142 Files changed 94 Nagoya talk: Graphs for performance are Ordered by used time per algorithm - gpusat quite good

Working with cmake remotly. ssh @(sg1.)dbai.tuwien.ac.at CPU branch wasn't working. Only AMD/Nvidia graphics with respective flags.

First steps: Connect ssh mkdir Ba cd Ba git clone https://github.com/mabseher/htd.git cd htd/ cmake . make -B -j1 make test (100pct tests passed, 0 tests failed out of 38 Total Test time (real) = 0.81 sec)

git clone https://github.com/VaeterchenFrost/GPUSAT.git

Configuration manual with the include options in CMakeLists.txt `https://github.com/VaeterchenFrost/GPUSAT/blob/master/CMakeLists.txt` or with help from `https://marketplace.visualstudio.com/items?itemName=ms-vscode.cmake-tools` to set up for the (potentially remote) environment.

4.1.1. Class Graphoutput

First steps in automatically visualizing the solving process with its tree decomposition. Outputs a .dot graph !!!EXAMPLE!!! with the bags and their solution nodes.

Additionally two functions to generate a Neo4J Cypher ***Link*** Query with:

- one graph representing the SAT formula and queries to construct incidence, dual and primal representations. output as satFile = "cypherSatFormula.txt"
- a graph representing the tree decomposition of the primal graph with it's bags containing variables. output as tdFile = "cypherTreedec.txt"

4.1.2. Class SolverVisualization

Umsetzung Special fork connecting the Visualization directly to the Solver. Utilizing the c++ library JsonCpp `https://github.com/open-source-parsers/jsoncpp` version 1.9.2 Usage: ./gpusat [OPTIONS]

Options: -h,–help Print this help message and exit -s,–seed INT number used to initialize the pseudorandom number generator -f,–formula TEXT path to the file containing the sat formula -d,–decomposition TEXT path to the file containing the tree decomposition –CPU run the solver on a cpu –NVIDIA run the solver on an NVIDIA device –AMD run the solver on an AMD device –weighted use weighted model count –noExp don't use extended exponents -v,–verbose print additional program information -p,–nopreprocess skips the preprocessing step for debugging and visualisation-purposes –dataStructure TEXT in array,combined,tree=combined data structure for storing the solution -m,–maxBagSize INT=-1 max size of a bag on the gpu -w,–combineWidth INT=20 maxi-

mum width to combine bags of the decomposition -g,–graph TEXT filename for saving the decomposition graph –visufile TEXT filename for saving the visualization file

Beispiel

4.2. Integration in dpdb

Programm Umsetzung Beispiel

5. Application and Images

6. Summary and Outline

What is achieved? What worked good, what bad?

References

[1] "Graph database use cases and solutions." `http://neo4j.com/use-cases`, August 2016. [Online; accessed 11-June-2020].

[2] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Cambridge: Cambridge University Press, 1 ed., 2012.

[3] A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar, "Evaluation of an mso-solver," *Proc. of ALENEX 2012*, 01 2012.

[4] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, (New York, NY, USA), p. 151–158, Association for Computing Machinery, 1971.

[5] M. Zisser, *Solving the #SAT problem on the GPU with dynamic programming and OpenCL*. 2018.

[6] N. Robertson and P. Seymour, "Graph minors. iii. planar tree-width," *Journal of Combinatorial Theory, Series B*, vol. 36, no. 1, pp. 49 – 64, 1984.

[7] J. K. Fichte, "Parameterized complexity and its applications in practice," May. Summer 2019 (May 6th – May 16th).

A. Images

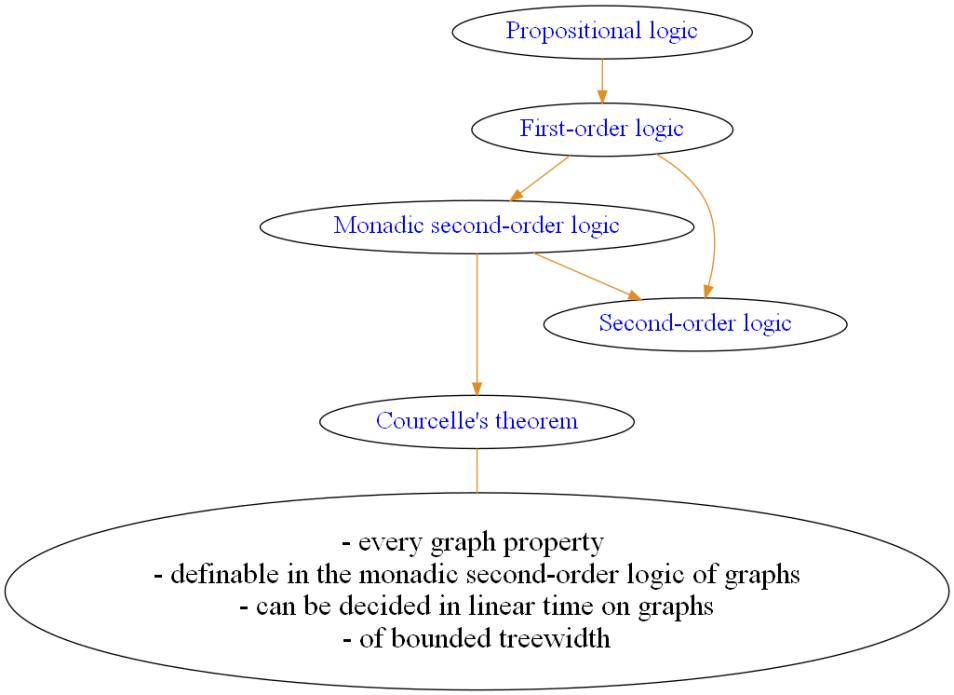

Figure 2: From propositional logic to monadic second order logic and Courcelle's Theorem