Advances in Efficient Probabilistic Reasoning with Answer Set Semantics

Thomas Eiter, Markus Hecher, Rafael Kiesel

Vienna University of Technology

funded by FWF project W1255-N23

5th of October 2021

Der Wissenschaftsfonds.

LOGICAL METHODS IN COMPUTER SCIENCE

Probabilistic Stream Reasoning

Interest in quantitative reasoning on top of Stream Reasoning:

Probabilistic Stream Reasoning

Interest in quantitative reasoning on top of Stream Reasoning:

- Probabilistic Reasoning [Nickles and Mileo, 2014]

Probabilistic Stream Reasoning

Interest in quantitative reasoning on top of Stream Reasoning:

- Probabilistic Reasoning [Nickles and Mileo, 2014]
- wLARS [Eiter and Kiesel, 2020] for
  - Probabilistic Reasoning
  - Preferential Reasoning
  - Algebraic Model Counting

Probabilistic Reasoning via Knowledge Compilation

- "Compile" the logical theory into a *tractable circuit representation* like d-DNNF or SDD

Probabilistic Reasoning via Knowledge Compilation

- "Compile" the logical theory into a *tractable circuit representation* like d-DNNF or SDD

  ↯ compilers like c2d [Darwiche, 2004] work on CNFs

Probabilistic Reasoning via Knowledge Compilation

- "Compile" the logical theory into a *tractable circuit representation* like d-DNNF or SDD

  ↯ compilers like c2d [Darwiche, 2004] work on CNFs

Two challenges:

Probabilistic Reasoning via Knowledge Compilation

- "Compile" the logical theory into a *tractable circuit representation* like d-DNNF or SDD

  ↯ compilers like c2d [Darwiche, 2004] work on CNFs

Two challenges:

- Logical connectives from the temporal domain

Probabilistic Reasoning via Knowledge Compilation

- "Compile" the logical theory into a *tractable circuit representation* like d-DNNF or SDD

  ↯ compilers like c2d [Darwiche, 2004] work on CNFs

Two challenges:

- Logical connectives from the temporal domain
- Answer set semantics

Probabilistic Reasoning via Knowledge Compilation

- "Compile" the logical theory into a *tractable circuit representation* like d-DNNF or SDD

  ↯ compilers like c2d [Darwiche, 2004] work on CNFs

Two challenges:

- Logical connectives from the temporal domain
- Answer set semantics

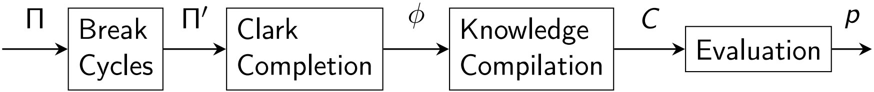
  ↪ Our recent work

Approach

- Start with normal answer set program Π

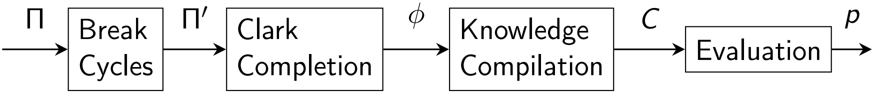
Approach

- Start with normal answer set program Π
- General strategy:



Approach

- Start with normal answer set program Π
- General strategy:



- **Goal:** Perform cycle breaking and Clark completion in such a way that compilation and evaluation are fast

Treewidth I

Definition (Tree decomposition, Treewidth)

Let G be a graph. Then a tree decomposition is a pair (T, χ), where T is a tree and χ is a labeling of $V(T)$ by subsets of $V(G)$ s.t.

- *for all nodes $v \in V(G)$ there is $t \in V(T)$ s.t. $v \in \chi(t)$;*
- *for every edge $\{v_1, v_2\} \in V(E)$ there exists $t \in V(T)$ s.t. $v_1, v_2 \in \chi(t)$;*
- *for all nodes $v \in V(G)$ the set of nodes $\{t \in V(T) \mid v \in \chi(t)\}$ forms a (connected) subtree of T.*

The width of (T, χ) is $\max_{t \in V'} |\chi(t)| - 1$. The treewidth of a graph is the minimal width of any of its tree decompositions.

Treewidth II

CNF:

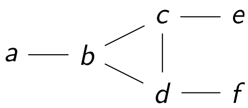
$$a \vee b$$
$$\neg b \vee c \vee d$$
$$\neg c \vee e$$
$$\neg d \vee f$$

Treewidth II

CNF:

$$a \lor b$$
$$\neg b \lor c \lor d$$
$$\neg c \lor e$$
$$\neg d \lor f$$

Graph:

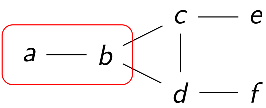


Treewidth II

CNF:

$$a \lor b$$
$$\neg b \lor c \lor d$$
$$\neg c \lor e$$
$$\neg d \lor f$$

Graph:

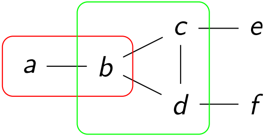


Treewidth II

CNF:

$$a \vee b$$
$$\neg b \vee c \vee d$$
$$\neg c \vee e$$
$$\neg d \vee f$$

Graph:

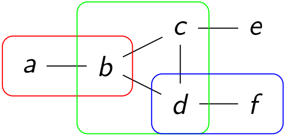


Treewidth II

CNF:

$$a \lor b$$
$$\neg b \lor c \lor d$$
$$\neg c \lor e$$
$$\neg d \lor f$$

Graph:

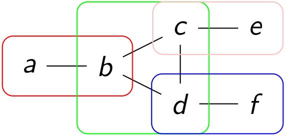


Treewidth II

CNF:

$$a \lor b$$
$$\neg b \lor c \lor d$$
$$\neg c \lor e$$
$$\neg d \lor f$$

Graph:

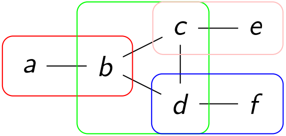


Treewidth II

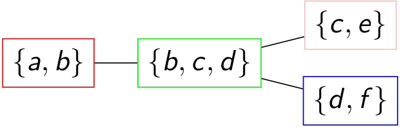
CNF:

$$a \lor b$$
$$\neg b \lor c \lor d$$
$$\neg c \lor e$$
$$\neg d \lor f$$

Graph:



Tree Decomposition:



Treewidth and Knowledge Compilation

- **Assumption:** Treewidth correlates with the time needed for knowledge compilation

Treewidth and Knowledge Compilation

- **Assumption:** Treewidth correlates with the time needed for knowledge compilation
- Why should that be true?

Treewidth and Knowledge Compilation

- **Assumption:** Treewidth correlates with the time needed for knowledge compilation
- Why should that be true?
- Knowledge compilation for a CNF ϕ with treewidth k feasible in $\mathcal{O}(|\phi|2^k)$ [Darwiche, 2004]

Treewidth and Knowledge Compilation

- **Assumption:** Treewidth correlates with the time needed for knowledge compilation
- Why should that be true?
- Knowledge compilation for a CNF ϕ with treewidth k feasible in $\mathcal{O}(|\phi|2^k)$ [Darwiche, 2004]
- There exist formulas ϕ with treewidth k s.t. the smallest SDD has size $\mathcal{O}(|\phi|2^k)$ [Amarilli *et al.*, 2018]

Treewidth and Knowledge Compilation

- **Assumption:** Treewidth correlates with the time needed for knowledge compilation
- Why should that be true?
- Knowledge compilation for a CNF ϕ with treewidth k feasible in $\mathcal{O}(|\phi|2^k)$ [Darwiche, 2004]
- There exist formulas ϕ with treewidth k s.t. the smallest SDD has size $\mathcal{O}(|\phi|2^k)$ [Amarilli *et al.*, 2018]
- Tree decomposition-based variable selection performs well [Korhonen and Järvisalo, 2021]

Cycle-Breaking Approaches

- We need a model-preserving cycle-breaking
  $\hookrightarrow$ disqualifies [Hecher, 2020], [Lin and Zhao, 2003]

Cycle-Breaking Approaches

- We need a model-preserving cycle-breaking
  $\hookrightarrow$ disqualifies [Hecher, 2020], [Lin and Zhao, 2003]
- Other cycle-breakings:

Cycle-Breaking Approaches

- We need a model-preserving cycle-breaking
  $\hookrightarrow$ disqualifies [Hecher, 2020], [Lin and Zhao, 2003]
- Other cycle-breakings:
  - lp2sat [Janhunen, 2004]:
    Resulting treewidth is $\mathcal{O}(k \cdot \log(|C|))$

Cycle-Breaking Approaches

- We need a model-preserving cycle-breaking
  $\hookrightarrow$ disqualifies [Hecher, 2020], [Lin and Zhao, 2003]
- Other cycle-breakings:
  - lp2sat [Janhunen, 2004]:
    Resulting treewidth is $\mathcal{O}(k \cdot \log(|C|))$
  - ProbLog [Mantadelis and Janssens, 2010]:
    Resulting treewidth is $\mathcal{O}(k \cdot 2^{|C|})$

where $|C|$ is the size of the largest strongly connected component (SCC) of the dependency graph of the program

Our Cycle-Breaking

- The largest SCC of the dependency graph may be large

Our Cycle-Breaking

- The largest SCC of the dependency graph may be large
- **Observation:** We can achieve a smaller increase when the *cyclicity* of the dependency graph is low

Our Cycle-Breaking

- The largest SCC of the dependency graph may be large
- **Observation:** We can achieve a smaller increase when the *cyclicity* of the dependency graph is low
- **Idea:** Split the strongly connected components into subgraphs of low cyclicity

Component-Boosted Backdoor Size

Definition ($\mathrm{cbs}(G)$)

Let G be a digraph. Then the component-boosted backdoor size of G, denoted $\mathrm{cbs}(G)$, is

- *1, if G is acyclic (which includes $V(G) = \emptyset$)*

Component-Boosted Backdoor Size

Definition ($\mathrm{cbs}(G)$)

Let G be a digraph. Then the component-boosted backdoor size of G, denoted $\mathrm{cbs}(G)$, is

- *1, if G is acyclic (which includes $V(G) = \emptyset$)*
- *2, if G is a polytree, i.e. the undirected version of G is connected and acyclic*

Component-Boosted Backdoor Size

Definition ($\mathrm{cbs}(G)$)

Let G be a digraph. Then the component-boosted backdoor size of G, denoted $\mathrm{cbs}(G)$, is

- *1, if G is acyclic (which includes $V(G) = \emptyset$)*
- *2, if G is a polytree, i.e. the undirected version of G is connected and acyclic*
- $\max\{\mathrm{cbs}(C) \mid C \in \mathsf{SCC}(G)\}$, *if G is cyclic but not strongly connected*

Component-Boosted Backdoor Size

Definition ($\mathrm{cbs}(G)$)

Let G be a digraph. Then the component-boosted backdoor size of G, denoted $\mathrm{cbs}(G)$, is

- *1, if G is acyclic (which includes $V(G) = \emptyset$)*
- *2, if G is a polytree, i.e. the undirected version of G is connected and acyclic*
- $\max\{\mathrm{cbs}(C) \mid C \in \mathrm{SCC}(G)\}$, *if G is cyclic but not strongly connected*
- $\min\{\mathrm{cbs}(G \setminus S) \cdot (|S| + 1) \mid S \subseteq V(G), S \neq \emptyset\}$ *otherwise*

Component-Boosted Backdoor Size

Definition ($\mathrm{cbs}(G)$)

Let G be a digraph. Then the component-boosted backdoor size of G, denoted $\mathrm{cbs}(G)$, is

- *1, if G is acyclic (which includes $V(G) = \emptyset$)*
- *2, if G is a polytree, i.e. the undirected version of G is connected and acyclic*
- $\max\{\mathrm{cbs}(C) \mid C \in \mathrm{SCC}(G)\}$, *if G is cyclic but not strongly connected*
- $\min\{\mathrm{cbs}(G \setminus S) \cdot (|S| + 1) \mid S \subseteq V(G), S \neq \emptyset\}$ *otherwise*

Intuitively, $\mathrm{cbs}(G)$ measures the *cyclicity* of G by decomposition into "easy to solve" subgraphs

Main Result

Theorem

For every answer set program Π, there exists an equivalent program Π' such that

1. *the answer sets are preserved bijectively*
2. *Π' is tight/acyclic*
3. *the treewidth of Π' is less or equal to $k \cdot \mathrm{cbs}(\mathsf{DEP}(\Pi))$, where k is the treewidth of Π.*

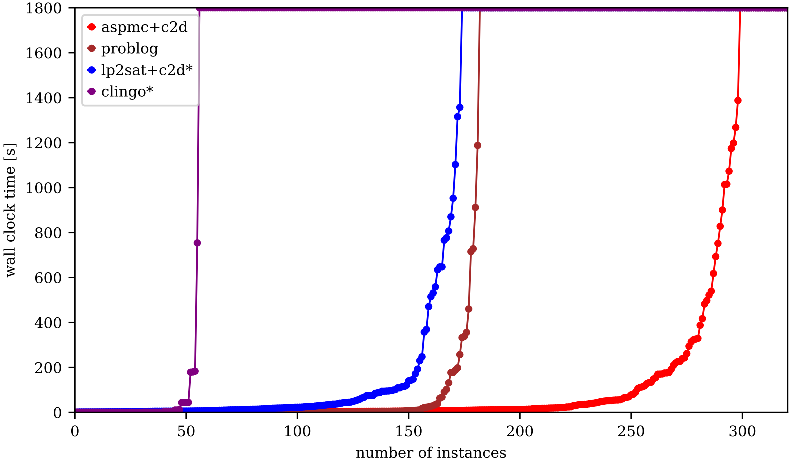
Scenarios

- S1 Probabilistic reasoning: Computing probabilities for atoms of Problog programs
- S2 Counting (small number of solutions on average): Counting the number of different paths between stations in public transport networks
- S3 Counting (many solutions on average): Counting conflict-free extensions in abstract argumentation

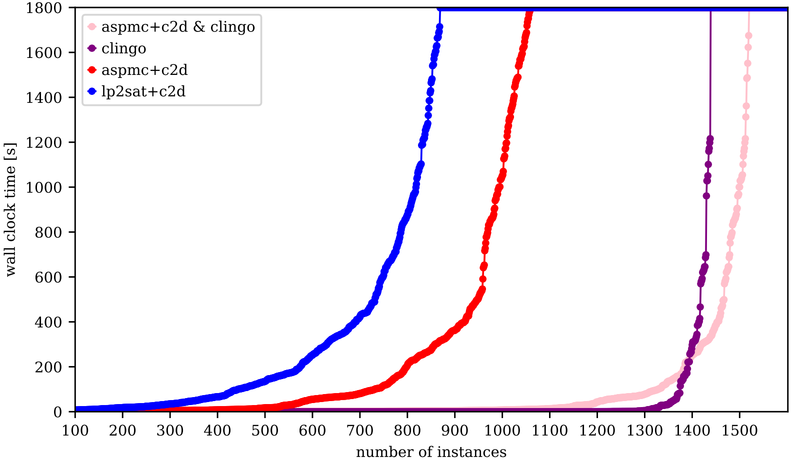
Solvers

- Problog, version 2.1.0.42, run with arguments "-k sdd"
- clingo, version 5.4.0, run with arguments "-q -n 0"
- lp2sat+c2d: cycle breaking due to [Bomanson, 2017] followed by compilation using c2d [Darwiche, 2004]
- aspmc+c2d: our cycle breaking followed by compilation using c2d [Darwiche, 2004]

Results S1


aspmc+c2d
problog
lp2sat+c2d*
clingo*
wall clock time [s]
number of instances
0
200
400
600
800
1000
1200
1400
1600
1800
0
50
100
150
200
250
300

Results S2


aspmc+c2d & clingo
clingo
aspmc+c2d
lp2sat+c2d
wall clock time [s]
number of instances
0
200
400
600
800
1000
1200
1400
1600
1800
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500

Results S3

solver configuration	$\sum$	tw ranges 0-300	300-600	>600	unique	time[h]
aspmc+c2d	**241**	**185**	**26**	**30**	**12**	**45.16**
lp2sat+c2d	182	182	0	0	0	73.85
clingo	144	97	21	26	2	94.78

Conclusions & Outlook

- Treewidth-awareness seems to be important for probabilistic reasoning

Conclusions & Outlook

- Treewidth-awareness seems to be important for probabilistic reasoning
- Improved approach for probabilistic reasoning under ASP semantics

Conclusions & Outlook

- Treewidth-awareness seems to be important for probabilistic reasoning
- Improved approach for probabilistic reasoning under ASP semantics
- Need to tackle the time domain!
  $\hookrightarrow$ Compile once and reuse: great!
  $\hookrightarrow$ Already hard for one timepoint, how problematic for more?

Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart.
Connecting knowledge compilation classes and width parameters.
arXiv preprint arXiv:1811.02944, 2018.

Jori Bomanson.
lp2normal - A normalization tool for extended logic programs.
In *LPNMR*, volume 10377 of *Lecture Notes in Computer Science*, pages 222–228. Springer, 2017.

Adnan Darwiche.
New advances in compiling CNF into decomposable negation normal form.
In *ECAI*, pages 328–332. IOS Press, 2004.

Thomas Eiter and Rafael Kiesel.
Weighted lars for quantitative stream reasoning.

In *Proc. ECAI'20*, 2020.

Markus Hecher.
Treewidth-aware Reductions of Normal ASP to SAT - Is Normal ASP Harder than SAT after All?
In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 485–495, 9 2020.

Tomi Janhunen.
Representing normal programs with clauses.
In *ECAI*, volume 16, page 358. Citeseer, 2004.

Tuukka Korhonen and Matti Järvisalo.
Integrating tree decompositions into decision heuristics of propositional model counters.

In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

Fangzhen Lin and Jicheng Zhao.
On tight logic programs and yet another translation from normal logic programs to propositional logic.
In *International Joint Conference on Artificial Intelligence*, 2003.

Theofrastos Mantadelis and Gerda Janssens.
Dedicated tabling for a probabilistic setting.
In *Technical Communications of the 26th International Conference on Logic Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

Matthias Nickles and Alessandra Mileo.

Web stream reasoning using probabilistic answer set programming.
In *International Conference on Web Reasoning and Rule Systems*, pages 197–205. Springer, 2014.