Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

The Impact of Structure in Answer Set Counting: Fighting Cycles and its Limits

Markus Hecher[1] , **Rafael Kiesel**[2]
[1] Massachusetts Institute of Technology, Cambridge, United States
[2] Vienna University of Technology, Vienna, Austria
hecher@mit.edu, rafael.kiesel@tuwien.ac.at

Abstract

Answer Set Programming is a widely used paradigm in knowledge representation and reasoning, which strongly relates to the satisfiability (SAT) of propositional formulas. While in the area of SAT, the last couple of years brought significant advances and different techniques for solving hard counting-based problems (e.g., #SAT, weighted counting, projected counting) that require more effort than deciding satisfiability, ASP still falls short. Intuitively, one explanation for this lies in the structure of a program, that – compared to SAT – was shown to yield strong evidence for being slightly less useful during solving. Indeed, for the structural measure treewidth that plays an important role in #SAT, ASP is expected to be at least slightly harder than SAT. The underlying source of this hardness increase lies in cyclic dependencies in the positive dependency graph. In this work, we consider which strategies are appropriate to tackle counting-based problems for ASP depending on cycle lengths. We present different encodings to counting-based variants of SAT that utilize recent advances. For small cycle lengths, we demonstrate a novel strategy based on feedback vertex sets. While medium cycle lengths leave room for future improvements, surprisingly, if cycles are significantly larger than structural dependencies (treewidth), we obtain a polynomial algorithm.

1 Introduction

Counting the satisfying interpretations of a logical theory is a notoriously hard problem but has especially recently gained relevance due to its relation to probabilistic reasoning (Raedt, Kimmig, and Toivonen 2007; Lee and Yang 2017; Eiter, Hecher, and Kiesel 2021), parameter learning in neuro-symbolic reasoning (Manhaeve et al. 2019; Skryagin et al. 2021; Yang, Ishay, and Lee 2020), knowledge compilation (Huang and Darwiche 2005; Darwiche 2004; Lagniez and Marquis 2017), and other fields.

While there have been significant advances in recent years for #SAT, i.e., model counting if the considered logical theory is a propositional formula, the question remains open how #ASP, i.e., the corresponding problem in the context of Answer Set Programming (ASP), should be tackled. Indeed, despite the availability of existing approaches for solving #ASP (Janhunen and Niemelä 2011; Fichte et al. 2017; Eiter, Hecher, and Kiesel 2021; Fichte et al. 2022), which were shown to be effective in some settings, their runtime guarantees do not imply efficient performance in general.

Conceptually, existing state-of-the-art techniques for counting #SAT exactly, which are based on knowledge compilation (Darwiche 2004; Lagniez and Marquis 2017), caching (Thurley 2006), or structural measures like treewidth (Korhonen and Järvisalo 2021), follow a unified goal, see, e.g., (Fichte, Hecher, and Hamiti 2021): Decompose the instance or find parts of the instance that can be solved individually, such that individual solution counts can be combined to obtain the count for the overall solution.

For ASP, the situation seems different. Indeed, there exists evidence that *treewidth*, which intuitively measures "treelikeness", is at least slightly less useful for #ASP (Hecher 2022). Further, the best known algorithm for #ASP, whose runtime is *fixed parameter tractable* with respect to the treewidth k is double exponential in k (Pichler et al. 2014), whereas for #SAT it is single exponential.

In this work, we take a closer look at the underlying source of this increased hardness of *answer set counting (#ASP)* for treewidth compared to #SAT, namely the positive cyclic dependencies between atomic variables introduced by the stable model semantics of ASP. These lead to the fact that any translation from ASP to SAT that does not make use of auxiliary variables is expected to have an exponential blow up in general (Lifschitz and Razborov 2006). Additionally, the typical strategy that does not make use of auxiliary variables adds so-called loop formulas (Lin and Zhao 2004b), which can lead to an unbounded increase of treewidth.

We tackle the cyclic dependencies by identifying different strategies that come with promising runtime and treewidth guarantees. Thereby, the relation of the length of cyclic dependencies (SCC size) compared to the treewidth of the program turns out to be a useful criterion to decide for appropriate strategies. Our main contributions are:

- We establish an overview and a characterization of different strategies, depending on cycle sizes (SCC sizes). We distinguish three different ranges of SCC sizes and we propose corresponding solving techniques.
- For small cycles, we present a new approach that combines two techniques from the literature (Janhunen and Niemelä 2011; Eiter, Hecher, and Kiesel 2021). Here, we consider the *feedback vertex set size (FVS size)*, which is strictly smaller than SCC size. This allows us to provide strictly better upper bounds combined with matching lower bounds under *ETH (exponential time hypothesis)*.

- Interestingly, the proposed approach for small cycles also works for medium-sized cycles. While a small gap remains open to our ETH-based lower bounds, we discuss the challenges associated with the closing of it and possible techniques for achieving this.
- Finally, we establish new strategies for large cycles. First, we show how to apply projected model counting (PMC) for counting answer sets. The second approach uses a novel encoding to #SAT that bijectively preserves the answer sets. This encoding even works for disjunctive programs; it is based on a novel approach that is single-exponential only in the program's treewidth. However, due to large cycles, the encoding is polynomial in the instance size. Further, we show that with our encoding answer set counting can be carried out in *polynomial time*.

Our work shows that while there are still gaps between the upper and lower bounds for #ASP with medium SCC size, we have efficient strategies for both small and large SCC size that cannot be significantly improved under ETH. Additionally, our novel encoding based on a fine grained cycle analysis that exploits FVS size pushes the limits of tractability in the presence of cycles even further.

Related Work. For disjunctive programs and extensions thereof, algorithms have been proposed (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017) running in time linear in the instance size, but double exponential in the treewidth. Under ETH, one cannot significantly improve this runtime, using a result (Lampis and Mitsou 2017) for QBFs with quantifier depth two and a standard reduction (Eiter and Gottlob 1995) from this QBF fragment to disjunctive ASP. The consistency of normal programs for treewidth is expected to be slightly harder than SAT (Hecher 2022). This even holds in case the largest SCC size is bounded (Fandinno and Hecher 2021). Also programs, where the number of even and/or odd cycles is bounded, have been analyzed (Lin and Zhao 2004a), which is orthogonal to largest SCC size.

2 Preliminaries

We assume familiarity with SAT and denote by $\#\mathrm{SAT}(\varphi)$ the *number of models* of a Boolean formula φ. Further, $\mathrm{PMC}(\varphi, A)$ for a set A of variables corresponds to the *projected model count* $|\{M \cap A \mid M \subseteq \mathsf{var}(\varphi), M \models \varphi\}|$.

Below, we give the background on ASP, graph representations of programs, and their associated structural parameters. An *answer set program* Π (Eiter, Ianni, and Krennwallner 2009) is a finite set of *rules* r of the form

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_n, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m, \quad (1)$$

where a_i, b_j, c_j are propositional atoms. Given a rule r, let

$$H_r = \{a_1, \ldots, a_k\}, \quad B_r^+ = \{b_1, \ldots, b_n\},$$
$$B_r^- = \{c_1, \ldots, c_m\}, \quad B_r = \{b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m\}.$$

A program Π is *normal* if $k = 1$ and *disjunctive*, otherwise.

By slight abuse of notation, we write $\leftarrow b_1, \ldots, b_n, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m$ for $\bot \leftarrow b_1, \ldots, b_n, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m, \mathrm{not}\, \bot$, where $\bot$ is a fresh atom not in Π.

Further, we allow *choice rules* $\{a\} \leftarrow B_r^+, B_r^-$ as a shorthand for the two rules $a \leftarrow B_r^+, B_r^-, \mathrm{not}\, na$ and $na \leftarrow B_r^+, B_r^-, \mathrm{not}\, a$, where na is a fresh propositional atom. We denote by $\mathcal{A}(\Pi)$ the set of atoms occurring in Π.

An *interpretation*, denoted $\mathcal{I}$, is a subset of $\mathcal{A}(\Pi)$; it *satisfies* an atom $a \in \mathcal{A}(\Pi)$ (resp. $\mathrm{not}\, a$ for $a \in \mathcal{A}(\Pi)$), written $\mathcal{I} \models a$ (resp. $\mathcal{I} \models \mathrm{not}\, a$), if $a \in \mathcal{I}$ (resp. $a \notin \mathcal{I}$). It satisfies Π (is a *model* of Π), if for each rule $r \in \Pi$ it holds that either $\mathcal{I} \models H_r$, i.e., there exists some $a \in H_r$ such that $\mathcal{I} \models a$, or $\mathcal{I} \not\models B_r$, i.e., there exists some $l \in B_r$ such that $\mathcal{I} \not\models l$. Furthermore, $\mathcal{I}$ is an *answer set* of Π if it is a $\subseteq$-minimal model of its *GL-reduct* (Gelfond and Lifschitz 1988) $\Pi^{\mathcal{I}} = \{H_r \leftarrow B_r^+ \mid r \in \Pi, B_r^- \cap \mathcal{I} = \emptyset\}$.

Schematic rules with variables $X, Y, \ldots$ are implicitly universally quantified and their semantics is given by grounding (instantiation) with concrete values (constants).

Example 1 (Reachability). *We consider a standard problem that can easily be modelled using cyclic programs but is harder to model with a Boolean formula, namely reachability in a directed graph, as a running example.*

$$i(X) \leftarrow t(X) \qquad \{t(X)\} \leftarrow v(X)$$
$$i(Y) \leftarrow i(X), t(X,Y) \qquad \{t(X,Y)\} \leftarrow e(X,Y)$$

Here, we guess for each vertex ($v(X)$), whether we take it ($t(X)$), and for each edge $e(X,Y)$, whether we take it ($t(X,Y)$). If a vertex x is taken, then it is included ($i(x)$). If a vertex x is included and edge x, y is taken, y is included.

By taking the complete graph over vertices in $\{0,1,2\}$, we can ground and simplify the program to obtain Π_g:

$$\{t(0)\} \qquad \{t(1)\} \qquad \{t(2)\}$$
$$i(0) \leftarrow t(0) \qquad i(1) \leftarrow t(1) \qquad i(2) \leftarrow t(2)$$
$$\{t(0,1)\} \qquad \{t(1,2)\} \qquad \{t(2,0)\}$$
$$\{t(0,2)\} \qquad \{t(2,1)\} \qquad \{t(1,0)\}$$
$$i(0) \leftarrow i(2), t(2,0) \qquad i(0) \leftarrow i(1), t(1,0)$$
$$i(1) \leftarrow i(2), t(2,1) \qquad i(1) \leftarrow i(0), t(0,1)$$
$$i(2) \leftarrow i(1), t(1,2) \qquad i(2) \leftarrow i(0), t(0,2)$$

We consider graphs and digraphs, using the following notation. The vertex- and edge-set of a (di)graph $G = (V, E)$ is denoted by $V(G)$ and $E(G)$, respectively. For $V \subseteq V(G)$ we let $G[V]$ be the (di)graph obtained by removing all vertices not in V from $V(G)$ (i.e. $V(G[V]) = V(G) \cap V$) and removing all edges which use a vertex not in V (i.e. $E(G[V]) = E(V) \cap V \times V$). Further, we define $G \setminus V$ as $G[V(G) \setminus V]$. Subgraph $C = G[V]$ is *strongly connected* if every vertex in C is reachable from any other vertex in C.

We denote by $\mathrm{SCC}(G)$ the set of *strongly connected components (SCCs)* of a digraph G, which are strongly connected subgraphs $G[V]$, where V is subset-maximal. Based on this, the *condensation* of G is the graph $(\mathrm{SCC}(G), E^*)$ such that there is an edge $(V, V') \in E^*$ if $V \neq V'$ and for some $v \in V, v' \in V$ there is an edge $(v, v') \in E(G)$.

The (positive) *dependency graph* D_Π of a program Π is the digraph G with $V(G)=\mathcal{A}(\Pi)$ and $(b, a) \in E(G)$ if there is a rule $r \in \Pi$ such that $a \in H_r$ and $b \in B_r^+$. The *primal graph* G_Π of Π is the graph G with $V(G)=\mathcal{A}(\Pi)$ and $(x, y) \in E(G)$ if there is a rule $r \in \Pi$ with $x, y \in H_r \cup B_r^+ \cup B_r^-$.

A *feedback vertex set* of a graph G is a set $F \subseteq V(G)$ such that (i) $G \setminus F$ is acyclic, i.e., there is no node that is reachable from itself, and (ii) F is minimal with respect to cardinality among the sets satisfying (i).

Example 2 (cont.)**.** *The dependency graph of Π_g is shown in Figure 1. It is the graph we used to ground the program, when reduced to the atoms of the form $i(x)$ for $x = 0, \ldots, 2$. The digraph has exactly one SCC $S = \{i(0), i(1), i(2)\}$ and thus its condensation is the digraph $(\{S\}, \emptyset)$.*

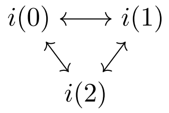
$$\begin{tikzcd}
i(0) \arrow[rr, leftrightarrow] \arrow[dr, leftrightarrow] & & i(1) \arrow[dl, leftrightarrow] \\
& i(2) &
\end{tikzcd}$$

Figure 1: Dependency graph of Π_g reduced to atoms of the form $i(x)$ for $x = 0, \ldots, 2$.

Next, we recall the definition of treewidth.

Definition 1. *Let G be a graph. A* tree decomposition (TD) *(Robertson and Seymour 1986) is a pair (T, χ), where T is a rooted tree,* $\mathrm{chld}(t)$ *is the* set of child nodes *for every node t in T, 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 $\{v_1, v_2\} \in E(G)$ 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.*

Intuitively, treewidth is a measure of the distance of a graph from a tree. It is motivated by the fact that many computationally hard problems are tractable on trees. Correspondingly, trees are the only graphs that have treewidth 1. For low treewidth it is often possible to generalize tractability results by decomposing problems into smaller subproblems using a TD witnessing the low width.

2.1 Binary Counter Cycle Breaking

Cycle breaking is a technique that allows the translation of normal programs to propositional formulas. We use the cycle breaking by (Janhunen and Niemelä 2011) as the basis for an improved cycle breaking. The idea here is based on the insight that for a given interpretation $\mathcal{I}$ and a program Π it is determined for each atom $a \in \mathcal{A}(\Pi)$ how many steps the shortest derivation of a in $\Pi^{\mathcal{I}}$ takes.

Example 3 (cont.)**.** *A standard simple way to represent the length of derivations in the program of our running example is to consider instead the graph given in Figure 2. The idea*

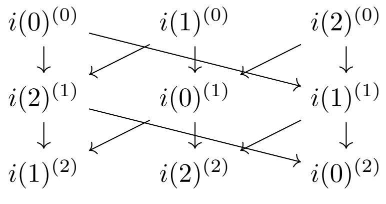
$$\begin{tikzcd}
i(0)^{(0)} \arrow[d] \arrow[rrd] & i(1)^{(0)} \arrow[d] \arrow[ld] & i(2)^{(0)} \arrow[d] \arrow[ld] \\
i(2)^{(1)} \arrow[d] \arrow[rrd] & i(0)^{(1)} \arrow[d] \arrow[ld] & i(1)^{(1)} \arrow[d] \arrow[ld] \\
i(1)^{(2)} & i(2)^{(2)} & i(0)^{(2)}
\end{tikzcd}$$

Figure 2: Unfolding of the dependency graph of Π_g reduced to atoms of the form $i(x)$ for $x = 0, \ldots, 2$.

is now the following: We can derive $i(1)^{(j)}$ iff we guess it to be true, $i(0)^{(j-1)}$ and $t(0,1)$ hold, or $i(2)^{(j-1)}$ and $t(2,1)$ hold. Thus, intuitively, the j in $i(x)^{(j)}$ tells us the number of steps used in this derivation. Note now that for a given guess $\mathcal{I}$ of the take atoms, if $i(x)$ is derived from the original program and $\mathcal{I}$, then there is a unique level j such that $i(x)^{(j)}$ holds but $i(x)^{(j-1)}$ does not (or $j = 0$).

The idea behind the translation of (Janhunen 2004) is now the following: introducing $|S(a)|$ copies per atom a, where $S(a)$ is the SCC of the dependency graph that contains a, is costly and should be avoided. Instead, each atom $a \in \mathcal{A}(\Pi)$ has an associated binary counter that can represent the numbers $0, \ldots, |S(a)| - 1$ via $\mathrm{c}(a) = \lceil \log_2(|S(a)|) \rceil$ auxiliary variables $\mathbf{b}_a^{(\mathrm{c}(a))}, \ldots, \mathbf{b}_a^{(1)}$. During cycle breaking, constraints are added to ensure that in every answer set $\mathcal{I}$ the number represented by the auxiliary variables $\mathbf{b}_a^{(\mathrm{c}(a))}, \ldots, \mathbf{b}_a^{(1)}$ corresponds to the length of the shortest derivation of a in $\Pi^{\mathcal{I}}$.

The basic cycle breaking proceeds by adding for each normal rule of the form (1) and atom $a \in \mathcal{A}(\Pi)$ additional rules:

$$a \leftarrow b_1, \ldots, b_n, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m. \tag{2}$$

That is, we keep all original rules.

$$just_a \leftarrow b_1, lt_{b_1,a}, \ldots, b_n, lt_{b_n,a}, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m \tag{3}$$

$$\leftarrow a, \mathrm{not}\, just_a \tag{4}$$

That is, we additionally require that if an atom a is true, then there needs to be a rule deriving it (i) whose body is true and (ii) the binary counter of every positive body literal b is less than the binary counter of a. This ensures that a derivation can only use positive literals that are derived strictly earlier.

$$next_a \leftarrow b_1, \ldots, b_n, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m, succ_{a,b_i} \quad \text{for } i = 1, \ldots, n \tag{5}$$

$$\leftarrow a, \mathrm{not}\, next_a \tag{6}$$

That is, we require that if an atom a is true, then there needs to be a rule deriving it (i) whose body is true and (ii) the binary counter of a is the successor of some positive body literal b. This means that there must be a rule deriving a which uses a positive body literal b whose derivation is one shorter than the one of a.

$$\leftarrow b_1, lt_{b_1,a}, \mathrm{not}\, succ_{a,b_1}, \ldots, b_n, lt_{b_n,a}, \mathrm{not}\, succ_{a,b_n}, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m \tag{7}$$

That is, we require that if a rule justifies the derivation of a, then (i) a has a shorter derivation than any derivation using this rule (if $\mathrm{not}\, lt_{b,a}$ holds for some positive body atom b) or (ii) the shortest derivation for a uses this rule (if $succ_{b,a}$ holds for some positive body atom b). This together with the two previous formulas enforces that the binary counter of a represents the length of its *shortest* derivation, rather than the length of some derivation.

Note here that $lt_{b,a}, succ_{a,b}$ are auxiliary variables whose truth value is defined in terms of the binary counters of a and b. For this, we use the following notation. Let:

- $S(.)$ be the function that assigns an atom a the SCC $S(a)$ of the dependency graph of Π such that $a \in S(a)$,
- $\succ_{top}$ be a topological ordering of the condensation of the dependency graph of Π.

We first define some auxiliary subformulas on the binary counters. For $a, b \in \mathcal{A}(\Pi)$ such that $S(a){=}S(b)$ and $1 \leq i \leq \mathrm{c}(a)$ we define auxiliary variables $[\mathbf{b}_a = \mathbf{b}_b]_{\geq i}$ via rules

$$[\mathbf{b}_a = \mathbf{b}_b]_{\geq \mathrm{c}(a)} \leftarrow \mathbf{b}_a^{(\mathrm{c}(a))}, \mathbf{b}_b^{(\mathrm{c}(a))}$$
$$[\mathbf{b}_a = \mathbf{b}_b]_{\geq \mathrm{c}(a)} \leftarrow \mathrm{not}\, \mathbf{b}_a^{(\mathrm{c}(a))}, \mathrm{not}\, \mathbf{b}_b^{(\mathrm{c}(a))}$$
$$[\mathbf{b}_a = \mathbf{b}_b]_{\geq i-1} \leftarrow \mathbf{b}_a^{(i-1)}, \mathbf{b}_b^{(i-1)}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq i}$$
$$[\mathbf{b}_a = \mathbf{b}_b]_{\geq i-1} \leftarrow \mathrm{not}\, \mathbf{b}_a^{(i-1)}, \mathrm{not}\, \mathbf{b}_b^{(i-1)}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq i}$$

and the auxiliary variables $[\mathbf{b}_a = \mathbf{b}_b + 1]_{\geq i}$ via the rules

$$[\mathbf{b}_a = \mathbf{b}_b + 1]_{\geq i} \leftarrow \mathbf{b}_a^{(j)}, \mathrm{not}\, \mathbf{b}_b^{(j)}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq j+1}, \mathrm{not}\, \mathbf{b}_a^{(i)}, \ldots, \mathrm{not}\, \mathbf{b}_a^{(j-1)}, \mathbf{b}_b^{(i)}, \ldots, \mathbf{b}_b^{(j-1)}$$

Based on this, the definition of "less than" and "successor" are given by the following rules:

$$\leftarrow \mathrm{not}\, a, \mathbf{b}_a^{(i)} \text{ for } i = 1, \ldots, \mathrm{c}(a),$$

and, if $S(a) \succ_{top} S(b)$

$$lt_{b,a} \leftarrow \qquad succ_{a,b} \leftarrow \mathrm{not}\, \mathbf{b}_a^{(1)}, \ldots, \mathrm{not}\, \mathbf{b}_a^{(\mathrm{c}(a))},$$

if $S(b) \succ_{top} S(a)$ we cannot derive $lt_{b,a}$ or $succ_{a,b}$, and if $S(a) = S(b)$ we add for $i = 1, \ldots, \mathrm{c}(a) - 1$

$$lt_{b,a} \leftarrow \mathbf{b}_a^{(i)}, \neg \mathbf{b}_b^{(i)}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq i+1}$$
$$succ_{a,b} \leftarrow [\mathbf{b}_a = \mathbf{b}_b + 1]_{\geq 1}$$

Note that although the definitions of $lt_{b,a}$, $succ_{a,b}$, $[\mathbf{b}_a = \mathbf{b}_b]_{\geq i}$, and $[\mathbf{b}_a = \mathbf{b}_b + 1]_{\geq i}$ are independent of the rule r that they are used in, we need to obtain a separate variable $v^{(r)}$ for each of the auxiliary variables v that is used only for this rule r, in order to prove our theoretical results. We refrain from adding the superscript (r) to each variable to reduce syntactic noise.

3 Which Cycles are Bad? – An Overview

While for ASP the complexity of the decision problem, in terms of treewidth has been established (Hecher 2022), for problems involving the *computation of more than one answer set*, e.g., counting all answer sets, there is *still a gap*. The reason for this gap is that existing approaches utilizing structural dependencies (treewidth) are based on computing local (relative) orderings on atom subsets, which are insufficient for bijectively preserving all the answer sets. Instead, these approaches then obtain duplicates, i.e., for one answer set, several solutions might be computed. As an alternative to reduce the cyclicity of a program, one could use *global orderings*, e.g., (Janhunen and Niemelä 2011), where atoms are ordered globally, thereby precisely preserving the answer sets. This approach, however, may arbitrarily *worsen the structure* in case of large cycles, since in the worst case the resulting treewidth is increased by a factor that is logarithmic in the largest cycle (SCC size). As a result, this increase is *not bounded in the treewidth* of the original program, since cycles might involve all the program's atoms. This is not surprising, as even current state-of-the-art ASP solvers involve the dynamic addition of constraints (cf. (Lin and Zhao 2004b)) over major parts of potential answer sets and are therefore not expected to be restricted to local parts.

The observations above motivate a deeper study on the different strategies for solving #ASP when considering the structure of programs and its largest SCC sizes. In the following, we classify the mentioned gap for #ASP, where we develop a range of cycle lengths (relative to treewidth k) with different general strategies for obtaining decent runtime bounds. We provide an overview of these ranges in Table 1. For *acyclic or mildly cyclic* programs, i.e., where SCC sizes are constant, we inherit ETH-tight runtime bounds from SAT, which yields runtimes of $2^{\Theta(k)} \cdot \mathsf{poly}(n)$. These cases are therefore (almost) identical to #SAT, where we obtain matching upper and lower bounds (under ETH). Then, for *non-constant* SCC sizes s that are polynomial in k, we obtain treewidth runtimes of the form $2^{\Theta(k \cdot \log(k))} \cdot \mathsf{poly}(n)$, which also matches known lower bounds.

For *super-polynomial* SCC sizes s we obtain treewidth dependencies of the form $2^{\Theta(k \cdot \log(s))}$. While we provide improvements for such SCC sizes, there is still a gap in the worst case. Interestingly, for *large SCC sizes s* that are at least double-exponential in k, we obtain actually polynomial runtimes. In other words, programs with SCC sizes that are superpolynomial in the treewidth and below double-exponential, are the challenging ones.

4 Coping with Small to Medium Cycles

We show how we can deal with small to medium length cycles by using and extending the approach of (Janhunen and Niemelä 2011). Our main insight here is that we can exploit a more fine grained structure of the cyclicity of the dependency graph than SCC size s, namely $|\mathbf{f}_S|$, the smallest feedback vertex set size (FVS size). While Janhunen and Niemelä's approach needs a binary counter with $\lceil \log_2(s) \rceil$ bits, for us $\lceil \log_2(|\mathbf{f}_S|) \rceil$ bits suffice. We show this reduction transferring to derived treewidth and runtime upper bounds.

The basis of our idea is to assign the binary counter the length of a shortest derivation, however considering a modified definition of length. Namely, instead of naïvely unfolding the graph as in Example 3, we consider an advanced unfolding technique called $\mathrm{T}_{\mathcal{P}}$-Unfolding from (Eiter, Hecher, and Kiesel 2021). Intuitively, it can exploit low cyclicity of the dependency graph to produce a more succinct unfolding.

We do so by using the observation that if the dependency graph is a polytree, then we can fix a root v arbitrarily, first consider all derivations towards the root and then consider all derivations away from the root. This means that using the strategy of copy atoms, we only need two copies for each atom. However, dependency graphs are not polytrees, thus, we need to reduce the general case to the case of polytrees.

To achieve this, we observe that if F is a FVS of the undirected version of the dependency graph D_Π, then $D_\Pi \setminus F$ is a polytree. Now, we can alternatingly consider all derivations of those atoms that are not in F and those that are in F. By doing this $|F| + 1$ times, we can be sure that we covered every derivation. To cover all derivations of atoms not in F we only need to consider derivations in a polytree. Thus, together with the previous observation this means that we only need to introduce $2 \cdot (|F| + 1)$ copies of the atoms that are not in F and $|F|$ atoms that are in F.

While our example graph has high cyclicity and $\mathrm{T}_{\mathcal{P}}$-Unfolding does not provide a benefit on it for that reason, it

Classification	SCC Size $s = \lvert S\rvert$	Solving Approach	Upper Bound	Lower Bound (ETH)
No Cycles	$s = 0$	TW-Aware Compl. (Hecher 2022)	$2^{\mathcal{O}(k)} \cdot \mathsf{poly}(n)$	$2^{o(k)} \cdot \mathsf{poly}(n)$
Tiny Cycles	$s = \mathcal{O}(1)$	TW-Aw. C. + Ords. (Janhunen 2004)	$2^{\mathcal{O}(k)} \cdot \mathsf{poly}(n)$	$2^{o(k)} \cdot \mathsf{poly}(n)$
Small Cycles	$s \in [\Omega(1), k^{\mathcal{O}(1)}]$	TW-Aw. C. + Orderings, **Section 4**	$\mathbf{2^{\mathcal{O}(k\cdot\log(\min(k,\lvert f_S\rvert)))} \cdot \mathsf{poly}(n)}$	$2^{o(k\cdot\log(k))} \cdot \mathsf{poly}(n)$
Medium Cycles	$s \in [k^{\Omega(1)}, 2^{2^{o(k)}}]$	TW-Aw. C. + Orderings, **Section 4**	$\mathbf{2^{\mathcal{O}(k\cdot\log(\lvert f_S\rvert))} \cdot \mathsf{poly}(n)}$	$2^{o(k\cdot\log(k))} \cdot \mathsf{poly}(n)$
Large Cycles**	$s = 2^{2^{\Omega(k)}}$	Disjunctive Encoding, **Section 5**	$\mathbf{\mathsf{poly}(n)}$	$\mathsf{poly}(n)$

Table 1: Classification of the complexity of #ASP according to the largest SCC size of the program's dependency graph. Π is a normal (HCF) program of size n, k is the treewidth of G_Π, S is the largest SCC of D_Π, and $\mathbf{f}_S$ is a smallest feedback vertex set of S. New contributions providing improvements over existing results are highlighted in bold-face. *: This construction immediately works for disjunctive programs.

still serves well to illustrate the idea behind $\mathrm{T}_\mathcal{P}$-Unfolding.

Example 4 (cont.). *As the set F we can choose $\{i(2)\}$, since the dependency graph reduced to the vertices $i(0), i(1)$ is a polytree. For the polytree we choose the root $i(1)$. Then, we start by covering all derivations of atoms that are not in F (i.e. that are not $i(2)$) by adding*

$$i(0)^{(0,0)} \leftarrow t(0)$$
$$i(1)^{(0,0)} \leftarrow t(1) \qquad i(1)^{(0,0)} \leftarrow i(0)^{(0,0)}, t(0,1)$$
$$i(0)^{(0,1)} \leftarrow t(0) \qquad i(0)^{(0,1)} \leftarrow i(1)^{(0,0)}, t(1,0)$$

Next we cover the derivations of $i(2)$ by adding the rules

$$i(2)^{(1,0)} \leftarrow t(2) \qquad i(2)^{(1,0)} \leftarrow i(0)^{(0,0)}, t(0,2)$$
$$i(2)^{(1,0)} \leftarrow i(0)^{(0,1)}, t(0,2) \quad i(2)^{(1,0)} \leftarrow i(1)^{(0,0)}, t(1,2)$$

Again we cover all derivations of atoms different from $i(2)$ by adding the rules

$$i(0)^{(1,2)} \leftarrow t(0) \qquad i(0)^{(1,2)} \leftarrow i(1)^{(0,0)}, t(1,0)$$
$$i(0)^{(1,2)} \leftarrow i(2)^{(1,0)}, t(2,0) \quad i(1)^{(1,0)} \leftarrow t(1)$$
$$i(1)^{(1,0)} \leftarrow i(0)^{(1,2)}, t(0,1) \quad i(1)^{(1,0)} \leftarrow i(2)^{(1,0)}, t(2,1)$$
$$i(0)^{(1,3)} \leftarrow t(0) \qquad i(0)^{(1,3)} \leftarrow i(1)^{(1,0)}, t(1,0)$$
$$i(0)^{(1,3)} \leftarrow i(2)^{(1,0)}, t(2,0)$$

This covers all derivations. Consider the dependency graph

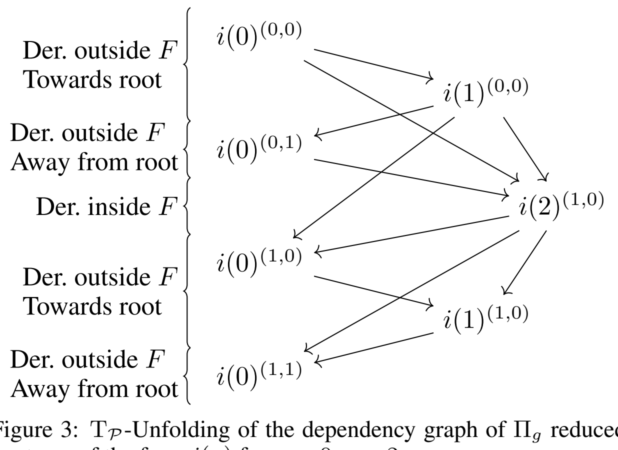

Figure 3: $\mathrm{T}_\mathcal{P}$-Unfolding of the dependency graph of Π_g reduced to atoms of the form $i(x)$ for $x = 0, \ldots, 2$.

of the $\mathrm{T}_\mathcal{P}$-Unfolded program in Figure 3. We see that we again introduce copies of atoms with a counter. However, contrary to before our counter is now a pair of values (n, f), where $n \in \mathbb{N}$ and $f \in \{0, 1\}$. The values signal the following: if we are considering derivations in the polytree and f is 0 the derivations are towards *the root; if f is 1, then we are considering derivations in the polytree* away from *the root. For vertices not in the polytree or the root of the polytree, $f = 1$ does not make sense, which is why we always assign $f = 0$ here. The second counter n tells us how many times we included atoms from F in any derivation branch.*

We define the enhanced cycle breaking formally. Assume that F_S is a feedback vertex set of S for each SCC S of the positive dependency graph of the given program Π. Then, we can make due with only $\mathrm{c}(i) = \lceil \log_2(|F_S| + 1) \rceil + 1$ bits for each atom a such that $S(a) = S$, as follows. Let:

- $S(.)$ be the function that assigns an atom a the SCC $S(a)$ such that $a \in S(a)$,
- $\succ_{top}$ be a topological ordering of the condensation of the dependency graph of Π,
- R_S be an arbitrary but fixed set of vertices from $S \setminus F_S$ such that every vertex in $S \setminus F_S$ has a path to exactly one vertex in R_S, and
- $\succ_S$ be a topological order induced by $S \setminus F_S$ when every edge is directed away from R_S.

Note that our cycle breaking only changes the definition of $lt_{b,a}$ and $succ_{a,b}$. Namely, $lt_{b,a}$ and $succ_{a,b}$ can still hold if the binary counters of a and b have the same value. Indeed, in our definition $succ_{a,b}$ only requires an increase of the counter when we are traveling from within the FVS to outside the FVS or from a vertex outside the FVS to a root. Conceptually, we model the two counters from Example 4 by using the least significant bit $\mathbf{b}_a^{(1)}$ to tell us whether were going towards a root or away from one (or whether we are in the FVS), and the rest of the bits to tell us how often we have visited the FVS any branch of our derivation.

Formally, we adapt the rules that define "less than" and "successor" to the following:

$$\leftarrow \mathrm{not}\, a, \mathbf{b}_a^{(i)} \text{ for } i = 1, \ldots, \mathrm{c}(a) \tag{8}$$

if $a \in F_{S(a)} \cup R_{S(a)}$

$$\leftarrow \mathbf{b}_a^{(i)} \text{ for } i = 1, \ldots, \mathrm{c}(a) \tag{9}$$

if $S(a) \succ_{top} S(b)$

$$lt_{b,a} \leftarrow \qquad succ_{a,b} \leftarrow \mathrm{not}\, \mathbf{b}_a^{(1)}, \ldots, \mathrm{not}\, \mathbf{b}_a^{(\mathrm{c}(a))} \tag{10}$$

if $S(b) \succ_{top} S(a)$ we cannot derive $lt_{b,a}$ or $succ_{a,b}$, but if $S(a) = S(b)$, we add for $i = 1, \ldots, \mathrm{c}(a) - 1$ the rules

$$lt_{b,a} \leftarrow \mathbf{b}_a^{(i)}, \neg \mathbf{b}_b^{(i)}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq i+1}. \qquad (11)$$

If in addition to $S(a) = S(b)$, we have $a \in F_{S(a)}$, we add

$$succ_{a,b} \leftarrow [\mathbf{b}_a = \mathbf{b}_b + 1]_{\geq 2}, \qquad (12)$$

or, if in addition $a \notin F_{S(a)}, b \in F_{S(a)}$, we add

$$lt_{b,a} \leftarrow succ_{a,b} \qquad succ_{a,b} \leftarrow [\mathbf{b}_a = \mathbf{b}_b]_{\geq 1} \qquad (13)$$

or, if in addition $a, b \notin F_{S(a)}, a \succ_{S(a)} b$, we add

$$lt_{b,a} \leftarrow succ_{a,b} \qquad succ_{a,b} \leftarrow \mathbf{b}_a^{(1)}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq 2} \qquad (14)$$

or, if in addition $a, b \notin F_{S(a)}, b \succ_{S(a)} a$, we add

$$lt_{b,a} \leftarrow succ_{a,b} \quad succ_{a,b} \leftarrow \text{not } \mathbf{b}_a^{(1)}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq 1} \quad (15)$$

As expected, we see that $lt_{b,a}$ and $succ_{a,b}$ hold if they hold for the original cycle breaking by Janhunen and Niemelä but they can hold more often, due to Rules (12)–(15).

Correctness, Treewidth-Awareness, and Runtimes. First, we establish correctness. Here, we denote by $Clark(\Pi_{FVS})$ the Clark Completion (Fages 1994) of the cycle breaking Π_{FVS} of a program Π, as specified in Rules (2)–(7) and (8)–(15).

Theorem 2 (Correctness)**.** *Let Π be a normal program. For every model $\mathcal{I}$ of Π there exists exactly one model $\mathcal{I}_{ext}$ of $Clark(\Pi_{FVS})$ such that $\mathcal{I}_{ext} \cap \mathcal{A}(\Pi) = \mathcal{I}$, and vice versa.*

This implies that the number of models of Π and $Clark(\Pi_{FVS})$ are equal, which implies that we can use $Clark(\Pi_{FVS})$ as a proxy for model counting of Π.

Proof (Sketch). We obtain the result by combining the results of (Janhunen and Niemelä 2011) and (Eiter, Hecher, and Kiesel 2021). (Eiter, Hecher, and Kiesel 2021) showed that $T_{\mathcal{P}}$-Unfolding satisfies the bijective preservation of models. Additionally, we can use the same strategy as in (Janhunen and Niemelä 2011) to show that the encoding $Clark(\Pi_{FVS})$ determines a unique extended $\mathcal{I}_{ext}$ such that $\mathcal{I}_{ext} \cap \mathcal{A}(\Pi) = \mathcal{I}$, if $\mathcal{I}$ is a model of Π but has no models $\mathcal{I}'$ such that $\mathcal{I}' \cap \mathcal{A}(\Pi)$ is not a model of Π. ☐

In order to establish treewidth-awareness we establish treewidth-awareness for the cycle breaking Π_{FVS}.

Theorem 3 (Treewidth-Awareness)**.** *Let Π be a normal program of treewidth k. Then, the treewidth of Π_{FVS} is in $\mathcal{O}(k \log(|\mathbf{f}_S|))$, where $|\mathbf{f}_S|$ is the FVS size.*

Note that we still need to be careful: by applying the standard version of Clark's Completion, we may observe an arbitrary increase of the treewidth. However, by making use of the ideas in (Hecher 2022), we can avoid this. Thus, treewidth-awareness follows for the whole translation from Π to $Clark(\Pi_{FVS})$, as long as Hecher's treewidth-aware version of Clark's Completion is used.

Proof (Sketch). Let $\mathcal{T} = (T, \chi)$ be a TD for Π of width k. We construct a TD for Π_{FVS} satisfying the desired bound.

The idea here is to add to every node $t \in T$ whose bag $\chi(t)$ contains a variable a all related auxiliary variables, i.e., $just_a, next_a, \mathbf{b}_a^{(i)}$. Since there are $\mathcal{O}(\log(|\mathbf{f}_S|))$ for every original variable a, this is okay.

Additionally, we however need to take care of the auxiliary variables $succ_{a,b}, lt_{b,a}, [\mathbf{b}_a = \mathbf{b}_b]_{\geq i}, [\mathbf{b}_a = \mathbf{b}_b + 1]_{\geq i}$. If we naïvely apply the same idea and add all of them, whenever $a, b \in \chi(t)$, the resulting TD has a width in $\Omega(k^2 \cdot \log(|\mathbf{f}_S|))$. Instead, we need to make use of one copy $v^{(r)}$ of each of these auxiliary variables v for each rule r such that $H_r = \{a\}$ and $b \in B_r^+$. Then, we can create copies $t^{(r)}$ for a node $t \in T$ and rule r such that $H_r \cup B_r \subseteq \chi(t)$ and add the variables for the rule r only to the bag of $t^{(r)}$. This way, $\chi(t^{(r)})$ only contains these auxiliary variables for a fixed a and r. Using this limitation, there are only $\mathcal{O}(k \log(|\mathbf{f}_S|))$ additional variables and the bound on the width holds. Further, by inspecting the rules, the constructed object is indeed a tree decomposition. ☐

Theorem 4 (Runtime for Normal Answer Set Counting)**.** *Let Π be a normal program such that the treewidth of G_Π is k and the FVS size of D_Π is $|\mathbf{f}_S|$. The number of answer sets of Π can be computed in time $2^{\mathcal{O}(k \cdot \log(|\mathbf{f}_S|))} \cdot \mathsf{poly}(\mathbf{n})$.*

Proof. First, we compute a TD of G_Π of width $5k$ in time $2^{\mathcal{O}(k)} \cdot (|\mathcal{A}(\Pi)| + |\Pi|)$ (Bodlaender et al. 2016). By Theorem 2, the reduction is correct, i.e., $\#\mathrm{ASP}(\Pi) = \#\mathrm{SAT}(\mathrm{Clark}(\Pi_{FVS}))$. By Theorem 3, the treewidth of Π_{FVS} is in $\mathcal{O}(k \cdot \log(|\mathbf{f}_S|))$ and by (Hecher 2022), we can modify Clark(.) to ensure that also the treewidth of $\mathrm{Clark}(\Pi_{FVS})$ is in $\mathcal{O}(k \cdot \log(|\mathbf{f}_S|))$.

Finally, we compute $\#\mathrm{SAT}(\mathrm{Clark}(\Pi_{FVS}))$ in time $2^{\mathcal{O}(k \cdot \log(|\mathbf{f}_S|))} \cdot \mathsf{poly}(|\mathcal{A}(\Pi)|)$ (Samer and Szeider 2010). ☐

4.1 The Challenge of Medium-Sized Cycles

While (Janhunen and Niemelä 2011)'s cycle breaking and our improved version of it come with good treewidth upper-bounds, the bounds are not only in terms of the program's treewidth but also have a logarithmic factor that depends on the cyclicity of the program. Thus, when program has low treewidth but high cyclicity, the treewidth of the produced SAT encodings is dominated by the logarithmic factor, rather than the treewidth of the program. E.g., for $|\mathbf{f}_S| = 2^{k^c}$, where k is the treewidth and c is a constant, the binary counter cycle breaking leads to a runtime upper bound of $2^{\mathcal{O}(k^{c+1})} \cdot \mathsf{poly}(|\mathcal{A}(\Pi)|)$, however, the lower bound of $2^{\Omega(k \cdot \log(k))} \cdot \mathsf{poly}(|\mathcal{A}(\Pi)|) = 2^{\Omega(k \cdot \log(k) + k^c)}$ shows a gap. This is undesirable, and poses the question whether there is an encoding into SAT that comes with a treewidth upper-bound of $\mathcal{O}(k^b)$, where k is the treewidth of the program and $b \in \mathbb{N}$ is a constant not depending on the cyclicity.

This question has been open for a while and work on the fixed parameter tractability of answer set program has only been able to prove that a *satisfiability* preserving encoding with a treewidth upper-bound of $\mathcal{O}(k \cdot \log(k))$ exists (Hecher 2022) but to the best of our knowledge no progress has been made for a *parsimonious* encoding.

On the other hand, it may be possible that such an encoding does not exist. In this section, we consider this possibility and recall a key result that may be useful in proving a lower bound. For this, we briefly explain current techniques (Wallon and Mengel 2020) for treewidth lower bounds of SAT encodings with auxiliary variables, which make use of size lower bounds for so-called *structured (d)-DNNFs* that are tractable circuit representations of Boolean functions (Darwiche and Marquis 2002).

Theorem 5. *Let f be a Boolean function over the set of variables X and C be a CNF with auxiliary variables Y such that $f \equiv \exists Y C$ and such that the treewidth of C is k. (i) Then there exists a structured DNNF D of size $|C| \cdot 2^{\mathcal{O}(k)}$ such that $D \equiv f$. (ii) If additionally C has the same number of satisfying assignments as f, then there exists a structured d-DNNF D of size $|C| \cdot 2^{\mathcal{O}(k)}$ such that $D \equiv f$.*

This implies that size lower bounds for structured DNNFs (resp. d-DNNFs) translate to treewidth lower bounds for satisfiability preserving (resp. parsimonious) encodings.

Proof (Sketch). (i) is known (Oztok and Darwiche 2017). For (ii), we observe that in this case the truth values of the variables in X are functionally determined by the assignment to the remaining variables in every model. This implies that the structured DNNF constructed in the proof by (Oztok and Darwiche 2017) is also a structured d-DNNF. □

Current techniques for lower-bounds use so-called *communication complexity* of the considered Boolean function.

Theorem 6 (Wallon and Mengel 2020)**.** *Let f be a Boolean function with non-deterministic (resp. deterministic) communication complexity cc. Then any DNNF (resp. d-DNNF) D such that $D \equiv f$ has size $\Omega(2^{cc})$.*

If we want to show that there is no parsimonious encoding for normal answer set programs that comes with a treewidth upper-bound of $2^{\mathcal{O}(k^b)}$, we need to use deterministic communication complexity. Namely, since there is a satisfiability preserving encoding with treewidth in $\mathcal{O}(k \cdot \log(k))$, any lower bound provided by non-deterministic communication complexity cannot be higher than $\mathcal{O}(k \cdot \log(k))$.

While this result may be of use, proving lower-bounds for the communication complexity of a function is a notoriously challenging task (Lee and Shraibman 2009). Additionally, we know from the previous section that a double exponential lower bound w.r.t. the treewidth of the program is not possible on a broad range of programs, i.e., those that have small SCCs or FVSs compared to their treewidth. This means that even if there is a family of programs, whose deterministic communication complexity grows exponentially in their treewidth, these programs need to be unusual.

Summarizing, both a better lower bound and a better upper bound are likely very hard to derive in this context. While we consider this problem a pressing matter, we therefore restrict ourselves to this short discussion of a possible point of attack for better lower bounds but leave it open.

5 Polynomial Encodings for Large Cycles

For programs with large cycles in its dependency graph, it turns out that one can count the number of answer sets in polynomial time. First, we present a reduction that counts answer sets via counting interpretations that are not models of the program, as well as by counting those models that are not answer sets. To this end, we utilize both $\#\mathrm{SAT}$ as well as projected model counting (PMC), which runs in polynomial time in the instance size due to the large cycles. Afterwards, we show a different approach that focuses on directly counting answer sets by means of a reduction to $\#\mathrm{SAT}$ that even works for disjunctive programs.

In the following, we assume a program Π that *might be disjunctive* and a tree decomposition $\mathcal{T} = (T, \chi)$ of G_Π of width k. Further, we let Π contain a *large cycle*, i.e., we let S be an SCC of D_Π such that $|S| \geq 2^{2^k}$. The reductions of the next two subsections carefully take care that the treewidth increase of the constructed Boolean formulas is bounded in order to guarantee polynomial-time solvability. For the ease of presentation we assume for every TD node t in T a *bag program* Π_t, which is a subset of $\{r \in \Pi \mid \mathcal{A}(r) \subseteq \chi(t)\}$ with $|\Pi_t| \leq 1$ and $\Pi = \bigcup_{t \text{ in } T} \Pi_t$. This is not a restriction, since it is achievable by adding auxiliary copy TD nodes such that each bag program contains (at most) one rule.

5.1 Counting Inverse Answer Sets via PMC

In order to ensure that the answer sets of Π containing large cycles, can be counted in polynomial time, one has to take care the treewidth increase is bounded. In the following, we design two reductions. One of these reductions constructs the formula F_{unsat}, whose models precisely capture those interpretations of Π that are not models of Π. The second reduction aims at constructing $F_{smaller}$, whose models restricted to $\mathcal{A}(\Pi)$ correspond to those models of Π, which are not answer sets of Π. Overall, these two formulas allow us to indirectly count the number of answer sets by computing $|2^{\mathcal{A}(\Pi)}| - \#\mathrm{SAT}(F_{unsat}) - \mathrm{PMC}(F_{smaller}, \mathcal{A}(\Pi))$.

Computing non-models of Π via formula F_{unsat}. For computing those interpretations of Π that are not models, we use variables $\mathcal{A}(\Pi)$, as well as $usat_t$ ($usat_{\leq t}$) for every node t in T to indicate that a rule in Π_t (a rule in $\Pi_{\leq t}$) is unsatisfied, respectively. Then, we define unsatisfiability of a rule in a node t by Formulas (16). If $\Pi_t = \emptyset$, this is not given by Formulas (17). Then, this information is propagated from nodes to parent nodes with the help of Formulas (18). Finally, to compute non-models, we enforce unsatisfiability up to the root node of T, by Formulas (19).

Define Unsatisfiability

$$usat_t \leftrightarrow \bigwedge_{a \in B_r^+} a \bigwedge_{b \in H_r \cup B_r^-} \neg b \qquad \text{for every } t \text{ in } T, \Pi_t = \{r\} \tag{16}$$

$$\neg usat_t \qquad \text{for every } t \text{ in } T, \Pi_t = \emptyset \tag{17}$$

$$usat_{\leq t} \leftrightarrow \bigvee_{t' \in \mathrm{chld}(t)} usat_{\leq t'} \vee usat_t \qquad \text{for every } t \text{ in } T \tag{18}$$

Enforce Unsatisfiability

$$usat_{\leq n} \qquad \text{for root node } n \text{ of } T \tag{19}$$

Example 5. *Consider the program $\Pi = \{a \vee b \leftarrow ; c \vee e \leftarrow d; d \vee e \leftarrow b\}$. Then, $\{a\}$, $\{b, e\}$ $\{b, c, d\}$ are answer sets of Π. Figure 4 (left) depicts the primal graph of Π and Figure 4 (right) shows a TD $\mathcal{T} = (T, \chi)$ of G_Π. Formula F_{unsat} is as follows: For t_1, we construct $\{usat_{t_1} \leftrightarrow d \wedge \neg c \wedge \neg e, usat_{\leq t_1} \leftrightarrow usat_{t_1}\}$; for t_2, we create $\{usat_{t_2} \leftrightarrow \neg a \wedge \neg b, usat_{\leq t_2} \leftrightarrow usat_{t_2}\}$; finally,*

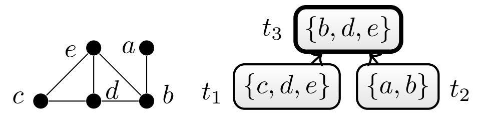

Figure 4: Primal graph G_Π (left) and a TD $\mathcal{T}$ of G_Π (right).

for t_3, this results in $\{usat_{t_3} \leftrightarrow b \wedge \neg d \wedge \neg e, usat_{\leq t_3} \leftrightarrow usat_{\leq t_1} \vee usat_{\leq t_2}, usat_{\leq t_3}\}$. It is easy to see that the models of F_{unsat} restricted to $\mathcal{A}(\Pi)$ are not models of Π; further, $\#\mathrm{SAT}(F_{unsat})$ gives the number of non-models of Π.

$F_{smaller}$**: Compute models of Π that are not answer sets.** For computing models of Π that are not answer sets, we use variables $\mathcal{A}(\Pi)$ as well as duplicates $\{\dot{a} \mid a \in \mathcal{A}(\Pi)\}$ (for computing reduct models). Further, for every node t in T, variable s_t ($s_{\leq t}$) indicates that the reduct model at node t (up to t) is strictly $\subseteq$-smaller than the model over variables $\mathcal{A}(\Pi)$, respectively. Then, Formulas (20) and (21) ensure that rules and reduct rules are satisfied, respectively. By Formulas (22), reduct models are $\subseteq$-smaller than the computed models over $\mathcal{A}(\Pi)$. Formulas (23) and (24) compute whether the reduct model is strictly $\subseteq$-smaller in t and up to t, respectively. Finally, Formula (25) ensures that the reduct model is strictly $\subseteq$-smaller than the model (up to the root n).

Satisfiability of Rules

$$\bigvee_{a \in B_r^+} \neg a \vee \bigvee_{b \in B_r^- \cup H_r} b \qquad \text{for every } r \in \Pi \qquad (20)$$

Define Satisfiability of Reduct

$$\bigvee_{a \in B_r^+} \neg \dot{a} \vee \bigvee_{b \in B_r^-} b \vee \bigvee_{c \in H_r} \dot{c} \qquad \text{for every } r \in \Pi \qquad (21)$$

Define $\subseteq$-Smaller Relation

$$\dot{a} \to a \qquad \text{for every } a \in \mathcal{A}(\Pi) \quad (22)$$

$$s_t \leftrightarrow \bigvee_{a \in \chi(t)} a \wedge \neg \dot{a} \qquad \text{for every } t \text{ in } T \qquad (23)$$

$$s_{\leq t} \leftrightarrow \bigvee_{t' \in \mathrm{chld}(t)} s_{\leq t'} \vee s_t \qquad \text{for every } t \text{ in } T \qquad (24)$$

Enforce Strict $\subseteq$-Inclusion

$$s_{\leq n} \qquad \text{for root node } n \text{ of } T \quad (25)$$

Correctness, Treewidth-Awareness, and Runtimes. First, we establish correctness as follows.

Lemma 7 (Correctness, $\star$[1])**.** *Let Π be a program and $\mathcal{T} = (T, \chi)$ be a TD of G_Π. Then, every model of formula F_{unsat} is not a model of Π, and vice versa. Further, every model M of $F_{smaller}$ when restricted to $\mathcal{A}(\Pi)$ corresponds to a model of Π such that there exists a model $M' \subsetneq M \cap \mathcal{A}(\Pi)$ that is a model of $\Pi^{M \cap \mathcal{A}(\Pi)}$ (and vice versa). As a result, $|2^{\mathcal{A}(\Pi)}| - \#\mathrm{SAT}(F_{unsat}) - \mathrm{PMC}(F_{smaller}, \mathcal{A}(\Pi))$ corresponds to the number of answer sets of Π.*

Indeed, the reductions linearly preserve treewidth.

Lemma 8 (Treewidth-Awareness)**.** *Let Π be a program and $\mathcal{T} = (T, \chi)$ be a TD of G_Π of width k. Then, the treewidths of $G_{F_{unsat}}$ and $G_{F_{smaller}}$ are linear in k.*

[1]Statements marked with "$\star$" are proven in the appendix.

Proof. Without loss of generality, we assume that $|\mathrm{chld}(t)| \leq 2$ for every node t in T, which can be achieved by adding $\mathcal{O}(|\mathcal{A}(\Pi)|)$ many intermediate copy nodes, resulting in a normalized (nice) representative of $\mathcal{T}$ (Kloks 1994)[Lem. 13.1.2]. Observe that both F_{unsat} and $F_{smaller}$ can be converted to CNF without additional auxiliary variables. Then, we construct a TD $\mathcal{T}' = (T, \chi')$ of $G_{F_{unsat}}$ and a TD $\mathcal{T}'' = (T, \chi'')$ of $G_{F_{smaller}}$, where χ', χ'' are constructed as follows. For every node t in T, we let $\chi'(t) = \chi(t) \cup \{usat_t, usat_{t'} \mid t' \in \mathrm{chld}(t)\}$. Further, for every t in T, we define $\chi''(t) = \chi(t) \cup \{\dot{a} \mid a \in \chi(t)\} \cup \{s_t, s_{\leq t}, s_{\leq t'} \mid t' \in \mathrm{chld}(t)\}$. Indeed, $\mathcal{T}'$ and $\mathcal{T}''$ is a TD of $G_{F_{unsat}}$ and $G_{F_{smaller}}$, respectively. Since $|\mathrm{chld}(t)| \leq 2$ for every node t in T, $|\chi'(t)|$ is in $\mathcal{O}(k)$. $\square$

These results yield the following runtime consequences.

Theorem 9 (Runtime for Answer Set Counting, $\star$)**.** *Let Π be a program such that the treewidth of G_Π is k and the largest SCC size of D_Π is $2^{2^{\Omega(k)}}$. Then, the number of answer sets of Π can be computed in time $\mathit{poly}(|\mathcal{A}(\Pi)|)$.*

Below we present a different method that has a similar runtime behavior, but even works for disjunctive programs.

5.2 Counting Answer Sets with $\#\mathrm{SAT}$

For programs with large cycles, it turns out that one can alternatively design a reduction to $\#\mathrm{SAT}$, running in polynomial time in the instance size. This approach has the advantage that indeed only one $\#\mathrm{SAT}$ solver call is enough to directly count the answer sets. While this reduction increases the treewidth exponentially (compared to the treewidth of the program's primal graph), it turns out that due to the large cycles, we can still count the satisfying assignments of the resulting formula in polynomial time in the program size.

For the ease of presentation, we additionally assume for every TD node t that there is a fresh auxiliary atom $\subset_t \in \chi(t)$. Greek letters α and β are constants that are available during the generation of the Boolean formula. So, expressions over these constants can be evaluated and simplified at formula compile time. These constants simplify the presentation and enable a compact notation of case distinctions.

In the reduction, we construct a formula F_{disj} using as variables the program atoms $\mathcal{A}(\Pi)$ as well as $usat_t^J$, $usat_{<t}^J$, and $usat_{\leq t}^J$ to indicate that interpretation $J \in 2^{\chi(t)}$ dissatisfies a rule $r \in \Pi_t$ in a node t below t, and up to t, respectively. To this end, we use auxiliary variables $usat_{<t,t'}^J$, with $t' \in \mathrm{chld}(t)$ as well as $comp^{J,K}$ to indicate compatibility of two assignments $J \in 2^{\chi(t)}$ and $K \in 2^{\chi(t')}$ for a child node $t' \in \mathrm{chld}(t)$. Then, the reduction uses auxiliary variables s_t^J to indicate that in node t, interpretation J is strictly $\subseteq$-smaller than the model over $\mathcal{A}(\Pi)$.

Formulas (26) take care that we compute models over variables $\mathcal{A}(\Pi)$. Then, Formulas (27) define whether $J \in 2^{\chi(t)}$ is strictly $\subseteq$-smaller than the model over $\mathcal{A}(\Pi)$. Further, Formulas (28) define compatibility between models $J \in 2^{\chi(t)}$ and $K \in 2^{\chi(t')}$ for a child node $t' \in \mathrm{chld}(t)$.

Unsatisfiability of J in node t is defined by Formulas (29), which is the case whenever J is not a model of the GL reduct of Π_t with respect to the interpretation over $\chi(t)$, or there is an atom $b \in J$ that is set to false in this interpretation (which implies that J is not even a subset). Then, Formulas (30) propagate this information to parent nodes by means of auxiliary Formulas (31) and (32).

In the end, Formulas (33) ensure that for the root node n of T, those assignments $J \in 2^{\chi(n)}$ that are strictly $\subseteq$-smaller (containing $\subset_n$) dissatisfy at least one rule in Π.

Satisfiability of Rules

$$\bigvee_{a \in B_r^+} \neg a \vee \bigvee_{b \in B_r^- \cup H_r} b \qquad \text{for every } r \in \Pi \tag{26}$$

Define Strict $\subseteq$-Smaller Reduct Interpretations

$$s_t^J \leftrightarrow \bigvee_{a \in \chi(t) \setminus J} a \qquad \text{for every } t \text{ in } T, J \in 2^{\chi(t)} \tag{27}$$

$$comp_t^{J,K} \leftrightarrow (\alpha \leftrightarrow (s_t^J \vee \beta)) \qquad \text{for every } t \text{ in } T, t' \in \text{chld}(t), J \in 2^{\chi(t)}, K \in 2^{\chi(t')}, J \cap \chi(t') = K \cap \chi(t), \alpha = (\subset_t \in J), \beta = (\subset_{t'} \in K) \tag{28}$$

Define Unsatisfiability of Reduct

$$usat_t^J \leftrightarrow \bigwedge_{\Pi_t = \{r\}, a \in B_r^-, b \in J} (\alpha \wedge \neg a) \vee \neg b \qquad \text{for every } t \text{ in } T, J \in 2^{\chi(t)}, \alpha = (J \not\models \{H_r \leftarrow B_r^+\}) \tag{29}$$

$$usat_{\leq t}^J \leftrightarrow (usat_{<t}^J \vee usat_t^J) \qquad \text{for every } t \text{ in } T, J \in 2^{\chi(t)} \tag{30}$$

$$usat_{<t,t'}^J \leftrightarrow \bigwedge_{\substack{K \in 2^{\chi(t')} \\ J \cap \chi(t') = K \cap \chi(t)}} (usat_{\leq t'}^K \vee \neg comp_t^{J,K}) \qquad \text{for every } t \text{ in } T, t' \in \text{chld}(t), J \in 2^{\chi(t)} \tag{31}$$

$$usat_{<t}^J \leftrightarrow \bigvee_{t' \in \text{chld}(t)} usat_{<t,t'}^J \qquad \text{for every } t \text{ in } T, J \in 2^{\chi(t)} \tag{32}$$

Enforce Unsatisfiability of Strictly $\subseteq$-Smaller Models

$$\alpha \rightarrow usat_{\leq n}^J \qquad \text{for root node } n \text{ of } T, J \in 2^{\chi(n)}, \alpha = (\subset_n \in J) \tag{33}$$

Correctness, Treewidth-Awareness, and Runtimes. The correctness demonstrates a bijective correspondence between models of the formula and answer sets of the program.

Lemma 10 (Correctness, $\star$)**.** *Let Π be a program and $\mathcal{T} = (T, \chi)$ be a TD of G_Π. Then, every model of formula F_{disj} restricted to $\mathcal{A}(\Pi)$ is an answer set of Π. Vice versa, every answer set of Π can be extended to a model of F_{disj}.*

This reduction cause an exponential increase of treewidth.

Lemma 11 (Treewidth-Awareness)**.** *Let Π be a program and $\mathcal{T} = (T, \chi)$ be a TD of G_Π of width k. Then, the treewidth of $G_{F_{disj}}$ is bounded by $2^{\mathcal{O}(k)}$.*

Proof. Without loss of generality, we assume $|\,\text{chld}(t) \leq 2|$ for every node t in T, achieved by adding $\mathcal{O}(|\mathcal{A}(\Pi)|)$ intermediate copy nodes (Kloks 1994)[Lem. 13.1.2]. Formula F_{disj} can be easily converted to CNF. We construct a TD $\mathcal{T}' = (T, \chi')$ of $G_{F_{disj}}$ where χ' is defined as follows. For every node t in T, we let $\chi'(t) = \chi(t) \cup \{s_t^J, comp_t^{J,K}, usat_t^{J \setminus \{\subset_t\}}, usat_{t'}^J, usat_{<t}^J, usat_{<t,t'}^J, usat_{\leq t}^J \mid t' \in \text{chld}(t), J \in 2^{\chi(t) \cup \{\subset_t\}}, K \in 2^{\chi(t') \cup \{\subset_{t'}\}}, J \cap \chi(t') = K \cap \chi(t)\}$. Indeed $\mathcal{T}'$ is a TD of $G_{F_{unsat}}$, i.e., variables of every instance of Formulas (26)–(33) are covered by a bag in $\mathcal{T}'$. Further, for every t in T, we have $|\chi(t')| \leq 2^{|\chi(t)|}$. As a result, the treewidth of $G_{F_{unsat}}$ is bounded by $2^{\mathcal{O}(k)}$. □

However, unfortunately one cannot significantly decrease this blowup, unless the exponential time hypothesis fails.

Proposition 12. *Let Π be a program and $\mathcal{T} = (T, \chi)$ be a TD of G_Π of width k. Then, unless ETH fails, there cannot be a reduction to a Boolean formula that runs in polynomial time in $\mathcal{A}(\Pi)$ such that the treewidth increase is in $2^{o(k)}$.*

Proof. Assume hat a polynomial-time reduction to SAT with treewidth increase in $2^{o(k)}$ exists. Then, we could decide the resulting formula in $2^{(2^{o(k)})} \cdot \mathsf{poly}(|\,\mathcal{A}(\Pi)|)$, which contradicts the known double-exponential lower bound for deciding consistency of Π (Fichte et al. 2017). □

These findings result in the following runtimes.

Theorem 13 (Polynomial Runtime for Answer Set Counting)**.** *Let Π be a program such that the width of G_Π is k and the largest SCC size of D_Π is $2^{2^{\Omega(k)}}$. Then, formula F_{disj} allows us to count the answer sets of Π in time $\mathsf{poly}(|\mathcal{A}(\Pi)|)$.*

6 Conclusion and Future Work

Our work provides new insights into the tractability of #ASP in terms of the structure (treewidth) of the rules. We saw that while it is limited by cyclic dependencies, oftentimes we are able to overcome the limitations by adapting to the SCC size in relation to its structural density (treewidth).

For low cyclicity, we provide a new approach for small cycles based on a more fine-grained structural analysis using feedback vertex sets. This approach reduces the problem to counting models of a propositional formula (#SAT), where significant advances have been presented recently (Korhonen and Järvisalo 2021; Soos and Meel 2022).

Additionally, we introduced new #SAT encodings for large cyclic dependencies, which surprisingly allow for a polynomial runtime. To the best of our knowledge, such a direct reduction that enables utilizing the efficiency of modern #SAT solvers has not been considered before.

While both approaches work for medium-sized cycles, unfortunately, our results leave a gap. However, we give compelling arguments why closing this gap is beyond challenging and outline a methodology we expect to be of use.

So, how hard are cycles for counting? Our theoretical findings might reveal phase transitions for #ASP from small over medium to large cycles. Indeed, we expect an easy-hard-easy pattern, since small cycles admit decent runtime guarantees that do not carry over to medium-sized cycles, while large cycles again come with polynomial runtime. For the future, we are interested in empirically comparing our new techniques for answer set counting to existing approaches. There, we also expect to observe such an easy-hard-easy pattern. For this, we are currently collecting and compiling a large set of reasonable benchmarks for #ASP, since competition instances do not capture challenges of quantitative approaches beyond consistency.

Acknowledgements

This work has been supported by the Austrian Science Fund (FWF) grants P32830, W1255-N23, and J4656, by the Society for Research Funding in Lower Austria (GFF, Gesellschaft für Forschungsfr̈derung NÖ) grant ExzF-0004, as well as by the Vienna Science and Technology Fund (WWTF) grant ICT19-065. Part of the work was carried out while Hecher was visiting the Simons Institute for the Theory of Computing.

References

Bodlaender, H. L.; Drange, P. G.; Dregi, M. S.; Fomin, F. V.; Lokshtanov, D.; and Pilipczuk, M. 2016. A $c^k n$ 5-Approximation Algorithm for Treewidth. *SIAM Journal on Computing* 45(2):317–378.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *J. Artif. Intell. Res.* 17:229–264.

Darwiche, A. 2004. New advances in compiling CNF into decomposable negation normal form. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, 328–332. IOS Press.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* 15(3–4):289–323.

Eiter, T.; Hecher, M.; and Kiesel, R. 2021. Treewidth-aware cycle breaking for algebraic answer set counting. In Bienvenu, M.; Lakemeyer, G.; and Erdem, E., eds., *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, 269–279.

Eiter, T.; Ianni, G.; and Krennwallner, T. 2009. Answer set programming: A primer. In *Reasoning Web International Summer School*, 40–110. Springer.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of logic in computer science* 1(1):51–60.

Fandinno, J., and Hecher, M. 2021. Treewidth-aware complexity in ASP: not all positive cycles are equally hard. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021*, 6312–6320. AAAI Press.

Fichte, J. K.; Hecher, M.; Morak, M.; and Woltran, S. 2017. Dynasp2. 5: Dynamic programming on tree decompositions in action. In *International Symposium on Parameterized and Exact Computation (IPEC)*, volume 89 of *LIPIcs*, 17:1–17:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Fichte, J. K.; Gaggl, S. A.; Hecher, M.; and Rusovac, D. 2022. IASCAR: incremental answer set counting by anytime refinement. In Gottlob, G.; Inclezan, D.; and Maratea, M., eds., *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, volume 13416 of *Lecture Notes in Computer Science*, 217–230. Springer.

Fichte, J. K.; Hecher, M.; Morak, M.; Thier, P.; and Woltran, S. 2023. Solving Projected Model Counting by Utilizing Treewidth and its Limits. *Artif. Intell.* 314:103810.

Fichte, J. K.; Hecher, M.; and Hamiti, F. 2021. The model counting competition 2020. *ACM J. Exp. Algorithmics* 26:13:1–13:26.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, 1070–1080. MIT Press.

Hecher, M. 2022. Treewidth-aware reductions of normal ASP to SAT - is normal ASP harder than SAT after all? *Artif. Intell.* 304:103651.

Huang, J., and Darwiche, A. 2005. DPLL with a trace: From SAT to knowledge compilation. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, 156–162. Professional Book Center.

Jakl, M.; Pichler, R.; and Woltran, S. 2009. Answer-set programming with bounded treewidth. In *IJCAI 2009, 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, volume 2, 816–822.

Janhunen, T., and Niemelä, I. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In Balduccini, M., and Son, T. C., eds., *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *Lecture Notes in Computer Science*, 111–130. Springer.

Janhunen, T. 2004. Representing normal programs with clauses. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, 358–362. IOS Press.

Kloks, T. 1994. *Treewidth. Computations and Approximations*, volume 842 of *LNCS*. Springer.

Korhonen, T., and Järvisalo, M. 2021. 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.

Lagniez, J., and Marquis, P. 2017. An improved decision-dnnf compiler. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 667–673. ijcai.org.

Lampis, M., and Mitsou, V. 2017. Treewidth with a quantifier alternation revisited. In *IPEC'17*, volume 89, 26:1–26:12. Dagstuhl Publishing.

Lee, T., and Shraibman, A. 2009. Lower bounds in communication complexity. *Found. Trends Theor. Comput. Sci.* 3(4):263–398.

Lee, J., and Yang, Z. 2017. Lpmln, weak constraints, and p-log. In Singh, S., and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 1170–1177. AAAI Press.

Lifschitz, V., and Razborov, A. A. 2006. Why are there so many loop formulas? *ACM Trans. Comput. Log.* 7(2):261–268.

Lin, F., and Zhao, X. 2004a. On odd and even cycles in normal logic programs. In McGuinness, D. L., and Ferguson, G., eds., *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 80–85. AAAI Press / The MIT Press.

Lin, F., and Zhao, Y. 2004b. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.* 157(1-2):115–137.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and Raedt, L. D. 2019. Deepproblog: Neural probabilistic logic programming. In Beuls, K.; Bogaerts, B.; Bontempi, G.; Geurts, P.; Harley, N.; Lebichot, B.; Lenaerts, T.; Louppe, G.; and Eecke, P. V., eds., *Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (Benelearn 2019), Brussels, Belgium, November 6-8, 2019*, volume 2491 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Oztok, U., and Darwiche, A. 2017. On compiling dnnfs without determinism. *CoRR* abs/1709.07092.

Pichler, R.; Rümmele, S.; Szeider, S.; and Woltran, S. 2014. Tractable answer-set programming with weight constraints: bounded treewidth is not enough. *Theory Pract. Log. Program.* 14(2):141–164.

Raedt, L. D.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2462–2467.

Robertson, N., and Seymour, P. D. 1986. Graph minors II: Algorithmic aspects of tree-width. *J. Algorithms* 7:309–322.

Samer, M., and Szeider, S. 2010. Algorithms for propositional model counting. *J. Discrete Algorithms* 8(1):50–64.

Skryagin, A.; Stammer, W.; Ochs, D.; Dhami, D. S.; and Kersting, K. 2021. SLASH: embracing probabilistic circuits into neural answer set programming. *CoRR* abs/2110.03395.

Soos, M., and Meel, K. S. 2022. Arjun: An efficient independent support computation technique and its applications to counting and sampling. In Mitra, T.; Young, E. F. Y.; and Xiong, J., eds., *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022*, 71:1–71:9. ACM.

Thurley, M. 2006. sharpsat - counting models with advanced component caching and implicit BCP. In Biere, A., and Gomes, C. P., eds., *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, 424–429. Springer.

Wallon, R., and Mengel, S. 2020. Revisiting graph width measures for cnf-encodings. *J. Artif. Intell. Res.* 67:409–436.

Yang, Z.; Ishay, A.; and Lee, J. 2020. Neurasp: Embracing neural networks into answer set programming. *29th International Joint Conference on Artificial Intelligence (IJCAI 2020).*