Characterizing Structural Hardness of Logic Programs: What Makes Cycles and Reachability Hard for Treewidth?

Markus Hecher

Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, United States
hecher@mit.edu

Abstract

Answer Set Programming (ASP) is a problem modeling and solving framework for several problems in KR with growing industrial applications. Also for studies of computational complexity and deeper insights into the hardness and its sources, ASP has been attracting researchers for many years. These studies resulted in fruitful characterizations in terms of complexity classes, fine-grained insights in form of dichotomy-style results, as well as detailed parameterized complexity landscapes. Recently, this lead to a novel result establishing that for the measure treewidth, which captures structural density of a program, the evaluation of the well-known class of normal programs is expected to be slightly harder than deciding satisfiability (SAT). However, it is unclear how to utilize this structural power of ASP. This paper deals with a novel reduction from SAT to normal ASP that goes beyond well-known encodings: We explicitly utilize the structural power of ASP, whereby we sublinearly decrease the treewidth, which probably cannot be significantly improved. Then, compared to existing results, this characterizes hardness in a fine-grained way by establishing the required functional dependency of the dependency graph's cycle length (SCC size) on the treewidth.

1 Introduction

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011; Gebser et al. 2012) is a declarative problem modeling and solving framework for knowledge representation and reasoning and artificial intelligence in general. This makes ASP a key formalism and suitable target language for solving problems in that area effectively, e.g., (Balduccini, Gelfond, and Nogueira 2006; Niemelä, Simons, and Soininen 1999; Nogueira et al. 2001; Guziolowski et al. 2013; Schaub and Woltran 2018; Abels et al. 2019). Such problems are thereby encoded in a logic program, which is a set of rules describing its solutions by means of so-called answer sets – an approach that goes beyond satisfying a set of clauses (rules) as in SAT, but additionally requires justifications for variables (atoms) that are claimed to hold. Considerable effort has been put into providing extensions and a rich modeling language that can be efficiently evaluated by solvers like clasp (Gebser, Kaufmann, and Schaub 2009) or wasp (Alviano et al. 2019).

Example 1 (Encoding with ASP)**.** *The classical way of encoding satisfiability (*SAT*) of a formula F is to guess for each variable $v \in \text{at}(F)$ its truth value via the two rules $v \leftarrow \neg\hat{v}$ and $\hat{v} \leftarrow \neg v$. Then, for every clause $l_1 \vee l_2 \vee \ldots \vee l_n$ in F, an additional constraint ensures the clause: $\bot \leftarrow \bar{l}_1, \bar{l}_2, \ldots, \bar{l}_n$ with $\bar{l}_i$ being v_i, if $l_i = \neg v_i$, and $\hat{l}_i$ otherwise.*

The computational complexity of ASP is fairly well studied, where for the *consistency problem* of deciding whether a so-called *normal logic program* admits an answer set is NP-complete (Bidoít and Froidevaux 1991; Marek and Truszczyński 1991). This result also extends to the class of *head-cycle-free (HCF)* programs (Ben-Eliyahu and Dechter 1994), but if full disjunctions are allowed in the heads of a rule, the complexity increases to Σ_2^P-completeness (Eiter and Gottlob 1995). Over the time, studying the complexity of ASP raised further attention. There is a wide range of more fine-grained studies (Truszczynski 2011) for ASP, also in the context of parameterized complexity (Cygan et al. 2015; Niedermeier 2006; Downey and Fellows 2013; Flum and Grohe 2006), where certain parameters (Lackner and Pfandler 2012) are taken into account. In parameterized complexity, the "hardness" of a problem is classified according to the effort required to solve the problem, e.g., runtime dependency, in terms of a certain *parameter*. For ASP there is growing research on the well-studied and prominent structural parameter treewidth (Jakl, Pichler, and Woltran 2009; Bichler, Morak, and Woltran 2020; Bliem et al. 2020). Intuitively, treewidth yields a *tree decomposition*, which is a structural representation that can be used for solving numerous combinatorially hard problems in parts; the treewidth indicates the maximum number of variables of these parts one has to investigate during problem solving.

Recently, it has been shown that when assuming the *Exponential Time Hypothesis (ETH)* (Impagliazzo, Paturi, and Zane 2001), which implies that SAT *cannot* be solved in time better than single-exponential in the number of variables in the worst case, normal ASP seems to be slightly harder (Hecher 2022) for treewidth than SAT. More precisely, (i) normal ASP can be solved in time $2^{\mathcal{O}(k \cdot \log(k))} \cdot \text{poly}(n)$ for any logic program of treewidth k with n variables (atoms) and under ETH this dependency on the treewidth can not be significantly improved. The reason for the hardness lies in very large cycles (*strongly connected components, SCCs*) of the program's dependency graph; the hardness proof re-

quires cycle lengths that are *unbounded in the treewidth*, i.e., cycles involve instance-size many atoms. Interestingly, this is in stark contrast to (ii) SAT, which can be decided in time $2^{\mathcal{O}(k)} \cdot \mathrm{poly}(n)$. The classical reduction of SAT to ASP, while preserving treewidth, does not introduce any cycles in the encoding (see Example 1). Thus, the question arises if one can construct cyclic programs to reduce SAT while decreasing treewidth. In more details, this paper asks:

- How can we encode SAT in (normal) ASP, thereby decreasing the treewidth by the amount that reflects the runtime difference between (i) and (ii)?
- Given the gap between unbounded cycle lenghts in (i) and no cycles in (ii), what is the difference in cycle length (SCC size) of the complexity between normal ASP and SAT? Can we bound the cycle length in the treewidth that still makes normal ASP hard?
- Can we draw further runtime consequences and lower bounds for other fragments or related extensions of ASP?

Contributions. We address these questions via a novel reduction that closes the gap to existing complexity results and lower bounds. Concretely, we provide the following results.

- First, we establish a novel reduction from SAT to normal ASP that in contrast to existing transformations fully utilizes the power of reachability along cycles, thereby *decreasing treewidth* from k to $\mathcal{O}(\frac{k}{\log(k)})$. Unless ETH fails, it is not expected that this reduction can be significantly improved, i.e., further major treewidth decreases are unexpected. To the best of our knowledge, this is the first reduction fully utilizing the structural power of normal ASP. Then, we also study the largest cycles (SCC sizes) of the dependency graph of the constructed program.
- Interestingly, the constructed cycles (SCC sizes) of the dependency graph are of size at most $2^{\mathcal{O}(k \cdot \log(k))}$. This is a major improvement compared to the largest SCC sizes of the recent hardness result, which is unbounded in the treewidth. Then, we show that for the class of ι-tight programs, the SCC sizes can be even decreased to $2^{\mathcal{O}(k \cdot \log(\iota))}$, while still preserving hardness for treewidth.
- Finally, we show that our reduction has immediate further implications in terms of computational complexity. We establish for the class of ι-tight programs a corresponding *ETH-tight lower bound*. Further, counting answer sets of a normal program with respect to a projection of interest is expected to be slightly harder than counting answer sets of disjunctive programs. Notably, both problems are complete for the same (classical) complexity class, but are surprisingly of different hardness for treewidth.

Related Work. Programs of bounded even or odd cycles have been analyzed (Lin and Zhao 2004). Further, the feedback width has been studied, which depends on the atoms required to break large SCCs (Gottlob, Scarcello, and Sideri 2002). There have been improvements for so-called ι-tight programs (Fandinno and Hecher 2021) with ι being smaller than treewidth k, which allow for runtimes of $2^{\mathcal{O}(k \cdot \log(\iota))} \cdot \mathrm{poly}(n)$. For normal and HCF programs, slightly superexponential algorithms in the treewidth (Fichte and Hecher 2019) for solving consistency are known. For disjunctive ASP algorithms have been proposed (Jakl, Pichler, and Woltran 2009; Pichler et al. 2014) running in time linear in the instance size, but double exponential in the treewidth. Hardness of further problems has been studied by means of runtime dependency in the treewidth, e.g., levels of exponentiality, where triple-exponential algorithms are known (Marx and Mitsou 2016; Fichte, Hecher, and Pfandler 2020).

Numerous reductions from ASP to SAT are known (Clark 1977; Ben-Eliyahu and Dechter 1994; Lin and Zhao 2003; Janhunen 2006; Alviano and Dodaro 2016; Bomanson and Janhunen 2013; Bomanson 2017). These reductions focus on the resulting formula size and number of auxiliary variables, where a sub-quadratic blow-up is unavoidable (Lifschitz and Razborov 2006). Unless ETH fails, a sub-quadratic blow-up in the treewidth cannot be circumvented as well (Hecher 2022). For SAT, empirical results (Atserias, Fichte, and Thurley 2011) involving resolution-width and treewidth yield efficient SAT solver runs on instances of small treewidth.

2 Preliminaries

We assume familiarity with graph terminology, cf., (Diestel 2012). Let $G = (V, E)$ be a directed graph. Then, a set $C \subseteq V$ of vertices of G is a *strongly-connected component (SCC)* of G if C is a $\subseteq$-largest set such that for every two distinct vertices u, v in C there is a directed path from u to v in G.

Tree Decompositions (TDs). A *tree decomposition (TD)* (Robertson and Seymour 1986) of a given graph $G{=}(V, E)$ is a pair $\mathcal{T}{=}(T, \chi)$ where T is a tree rooted at $\mathrm{root}(T)$ and χ assigns to each node t of T a set $\chi(t) \subseteq V$, called *bag*, such that (i) $V = \bigcup_{t \text{ of } T} \chi(t)$, (ii) $E \subseteq \{\{u, v\} \mid t \text{ in } T, \{u, v\} \subseteq \chi(t)\}$, and (iii) for each r, s, t of T, such that s lies on the path from r to t, we have $\chi(r) \cap \chi(t) \subseteq \chi(s)$. For every node t of T, we denote by $\mathrm{chldr}(t)$ the *set of child nodes of* t in T. We let $\mathrm{width}(\mathcal{T}) := \max_{t \text{ of } T} |\chi(t)| - 1$. The *treewidth* $tw(G)$ of G is the minimum $\mathrm{width}(\mathcal{T})$ over all TDs $\mathcal{T}$ of G. For a node t of T, we say that $\mathrm{type}(t)$ is *leaf* if t has no children; *join* if t has exactly two children t' and t'' with $t' \neq t''$; *inner* if t has a single child. If for every node t of T, $\mathrm{type}(t) \in \{\textit{leaf}, \textit{join}, \textit{inner}\}$, the TD is called *nice*. A TD can be turned into a nice TD (Kloks 1994)[Lem. 13.1.3] *without width-increase* in linear time. Without loss of generality, we assume that *bags of nice TDs are distinct*.

Answer Set Programming (ASP). We assume familiarity with propositional satisfiability (SAT) (Biere et al. 2009; Kleine Büning and Lettman 1999), where we use clauses, formulas, and assignments in the usual meaning. Two assignments $I : X \to \{0, 1\}$, $I' : X' \to \{0, 1\}$ are *compatible*, whenever for every $x \in X \cap X'$ we have that $I(x) = I'(x)$.

We follow standard definitions of propositional ASP (Brewka, Eiter, and Truszczyński 2011; Janhunen and Niemelä 2016). Let ℓ, m, n be non-negative integers such that $\ell \leq m \leq n$, a_1, ..., a_n be distinct propositional atoms. Moreover, we refer by *literal* to an atom or the negation thereof. A *program* Π is a set of *rules* of the form $a_1 \vee \cdots \vee a_\ell \leftarrow a_{\ell+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$. For a rule r, we let $H_r := \{a_1, \ldots, a_\ell\}$, $B_r^+ := \{a_{\ell+1}, \ldots, a_m\}$,

and $B_r^- := \{a_{m+1}, \ldots, a_n\}$. We denote the sets of *atoms* occurring in a rule r or in a program Π by $\mathrm{at}(r) := H_r \cup B_r^+ \cup B_r^-$ and $\mathrm{at}(\Pi) := \bigcup_{r \in \Pi} \mathrm{at}(r)$. Program Π is *normal* if $|H_r| \leq 1$ for every $r \in \Pi$. The *dependency graph* D_Π of Π is the directed graph defined on the atoms from $\bigcup_{r \in \Pi} H_r \cup B_r^+$, where for every rule $r \in \Pi$ two atoms $a \in B_r^+$ and $b \in H_r$ are joined by an edge (a, b).

An *interpretation* I is a set of atoms. I *satisfies* a rule r if $(H_r \cup B_r^-) \cap I \neq \emptyset$ or $B_r^+ \setminus I \neq \emptyset$. I is a *model* of Π if it satisfies all rules of Π, in symbols $I \models \Pi$. For brevity, we view propositional formulas as sets of formulas (e.g., clauses) that need to be satisfied, and use the notion of interpretations, models, and satisfiability analogously. The *Gelfond-Lifschitz (GL) reduct* of Π under I is the program Π^I obtained from Π by first removing all rules r with $B_r^- \cap I \neq \emptyset$ and then removing all $\neg z$ where $z \in B_r^-$ from the remaining rules r (Gelfond and Lifschitz 1991). I is an *answer set* of a program Π, denoted $I \models \Pi$, if I is a minimal model of Π^I. The problem of deciding whether an ASP program has an answer set is called *consistency*, which is Σ_2^{P}-complete (Eiter and Gottlob 1995). If the input is restricted to normal programs, the complexity drops to NP-complete (Marek and Truszczyński 1991).

The following characterization of answer sets is often invoked for normal programs (Lin and Zhao 2003). Let $A \subseteq \mathrm{at}(\Pi)$ be a set of atoms. Then, a function $\varphi : A \to \{0, \ldots, |A| - 1\}$ is an *ordering* over $\mathrm{dom}(\varphi) := A$. Let I be a model of a normal program Π and φ be an ordering over I. An atom $a \in I$ is *proven* if there is a rule $r \in \Pi$ *proving* a, where $a \in H_r$ with (i) $B_r^+ \subseteq I$, (ii) $I \cap B_r^- = \emptyset$ and $I \cap (H_r \setminus \{a\}) = \emptyset$, and (iii) $\varphi(b) < \varphi(a)$ for every $b \in B_r^+$. Then, I is an *answer set* of Π if (i) I is a model of Π, and (ii) I *is proven*, i.e., every $a \in I$ is proven. For an ordering φ and two atoms $a, b \in \mathrm{at}(\Pi)$, we write $a \prec_\varphi b$ whenever b *directly succeeds* a, i.e., $\varphi(b) = \varphi(a) + 1$. The empty ordering φ with $\mathrm{dom}(\varphi) = \emptyset$ is abbreviated by $\varnothing$.

Primal Graph. We need graph representations to use treewidth for ASP (Jakl, Pichler, and Woltran 2009). The *primal graph* $\mathcal{G}_\Pi$ of program Π has the atoms of Π as vertices and an edge $\{a, b\}$ if there exists a rule $r \in \Pi$ and $a, b \in \mathrm{at}(r)$. The primal graph $\mathcal{G}_F$ of a Boolean Formula F (in CNF) uses variables of F as vertices and adjoins two vertices a, b by an edge, if there is a clause in F containing a, b. Let $\mathcal{T} = (T, \chi)$ be a TD of $\mathcal{G}_F$. Then, for every node t of T, we define the *bag clauses* $F_t := \{c \in F \mid \mathrm{at}(c) \subseteq \chi(t)\}$.

Example 2. *Consider formula* $F := \{c_1, c_2, c_3\}$*, where* $c_1 := (a \vee \neg b)$*,* $c_2 := (\neg a \vee c \vee d)$*,* $c_3 := (\neg c \vee \neg d)$*. Figure 1 (left) depicts the primal graph* $\mathcal{G}_F$ *and Figure 1 (right) shows a TD of* $\mathcal{G}_F$*. Then, observe that* $F_{t_1} = \{c_1\}$*,* $F_{t_2} = \{c_2, c_3\}$*, and* $F_{t_3} = \emptyset$*.*

ι-Tightness. For a program Π and an atom $a \in \mathrm{at}(\Pi)$ we denote the *SCC of atom* a in D_Π by $\mathrm{scc}(a)$. Then, given a TD $\mathcal{T} = (T, \chi)$ of $\mathcal{G}_\Pi$, the *tightness width* is $\max_{t \text{ of } T} \max_{x \in \chi(t)} |\chi(t) \cap \mathrm{scc}(x)|$. The *tightness treewidth* ι of Π is the smallest tightness width among every TD of width in $\mathcal{O}(tw(\mathcal{G}_\Pi))$; in this case we say Π is *ι-tight*.

Proposition 1 ((Fandinno and Hecher 2021))**.** *Assume a normal, ι-tight program Π; the treewidth of $\mathcal{G}_\Pi$ is k. Then, consistency of Π can be decided in time* $2^{\mathcal{O}(k \cdot \log(\iota))} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$.

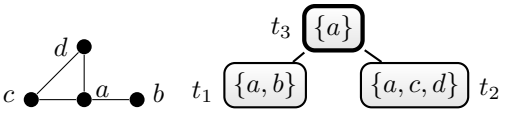

Figure 1: Graph G (left) and a TD $\mathcal{T}$ of G (right).

3 Decreasing Treewidth of SAT via ASP

In this section we show how to translate a Boolean formula into a logic program, thereby decreasing the treewidth and explicitly utilizing the structural power of ASP. Thereby, in contrast to the standard translation as sketched in Example 1, we explicitly utilize cycles and the power of reachability the ASP formalism provides.

The concrete decrease of treewidth of our reduction of the next subsection will be tightly linked to the following observation, which expresses that the factorial $k!$ of a number $k \in \mathbb{N}$ is bounded from below by $k^{\Omega(k)}$.

Observation 1. *Let $k \in \mathbb{N}$. Then, $k!$ is in $2^{\Omega(k \cdot \log(k))}$.*

Proof. We have $\frac{k!}{k^k} = e^{-\mathcal{O}(k)}$ by using Stirling's formula, see, e.g., (Lokshtanov, Marx, and Saurabh 2011). As a result, we derive that $k!$ corresponds to $\frac{2^{k \cdot \log(k)}}{e^{\mathcal{O}(k)}} = \frac{2^{k \cdot \log(k)}}{2^{\log(e) \cdot \mathcal{O}(k)}} = 2^{k \cdot \log(k) - \log(e) \cdot \mathcal{O}(k)} = 2^{\Omega(k \cdot \log(k))}$. □

This observation immediately implies that $k!$ is of the same order of magnitude as $k^{\Theta(k)}$, as obviously $k!$ is in $k^{\mathcal{O}(k)}$.

Decreasing Treewidth by the Power of Reachability

The idea of our main reduction $\mathcal{R}$ is as follows. We take an instance F of SAT, i.e., a Boolean formula and a nice tree decomposition $\mathcal{T} = (T, \chi)$ of $\mathcal{G}_F$ of width k. Then, we simulate for each node of T, the up to 2^k many assignments via $k'!$ many orderings, where k' shall be sufficiently smaller than k. More precisely, we decrease the treewidth from k to k' such that $k'! \geq 2^k$. Then, since $k'!$ is in $2^{\Omega(k' \cdot \log(k'))}$ (see Observation 1), we have that $2^{\Omega(k' \cdot \log(k'))}$ is at least 2^k and therefore $k' = \mathcal{O}(\frac{k}{\log(k')}) = \mathcal{O}(\frac{k}{\log(k)})$. As a result, our approach allows us to slightly reduce treewidth, thereby efficiently utilizing the power of ASP and positive cycles in order to solve SAT with less structural overhead. While this seems surprising, it is in line with the known hardness result of ASP, cf., Proposition 1 for $\iota = k$.

Formally, we determine k' by taking the smallest integer k' such that $k'! \geq 2^k$. We define such a value for every node t of T, where k'_t is the *smallest integer* such that $k'_t! \geq 2^{|\chi(t)|}$. Then, we define a set V_t of *ordering vertices* consisting of k'_t many fresh vertices that are uniquely determined by the bag $\chi(t)$, i.e., for any TD nodes t, t' we have $\chi(t) = \chi(t')$ if and only if $V_t = V_{t'}$. Using this set V_t, we refer to the resulting *set of at least* $2^{|\chi(t)|}$ *many orderings* among elements in V_t by $\mathrm{ord}(t)$. Further, for every node t of T, we refer to the bijective *mapping from a subset* $X \subseteq \mathrm{ord}(t)$ *of orderings to assignments* by $\mathcal{I}_t : X \to 2^{\chi(t)}$. More precisely, for every node t and ordering $\varphi \in \mathrm{ord}(t)$, the corresponding *unique assignment of φ over atoms in* $\chi(t)$ is given by $\mathcal{I}_t(\varphi)$ (if exists). Note that $\mathcal{I}_t$ is any arbitrary, but fixed bijection, i.e., it might be undefined for some unused orderings in $\mathrm{ord}(t)$.

Example 3. *Recall formula F and TD $\mathcal{T} = (T, \chi)$ of Example 2. By definition, we require that $|V_{t_1}|! \geq 2^2$, $|V_{t_2}|! \geq 2^3$, and $|V_{t_3}|! \geq 2$. As a result, we need to choose $|V_{t_1}| = 3$, $|V_{t_2}| = 4$, and $|V_{t_3}| = 2$. Consequently, there are orderings $\alpha \in \mathrm{ord}(t_1)$, $\beta \in \mathrm{ord}(t_2)$, where $\mathcal{I}_{t_1}(\alpha)$ and $\mathcal{I}_{t_2}(\beta)$ is not defined. By convention, we refer to the elements in V_{t_1} by v_1^j, to those in V_{t_2} by v_2^j and to those in V_{t_3} by v_3^j.*

Ordering-Augmented Tree Decompositions. Let $\mathcal{T} = (T, \chi)$ be a nice tree decomposition of $\mathcal{G}_\Pi$. In order to decouple the $k'!$ many assignments, we need to get access to any of the simulated orderings one-by-one. To this end, we define an *ordering-augmented tree decomposition*.

Definition 1. *Let F be a Boolean formula and $\mathcal{T} = (T, \chi)$ be a nice TD of $\mathcal{G}_F$. Then, we construct an ordering-augmented TD $\mathcal{T}' = (T', \chi', \varphi, \psi)$ of $\mathcal{G}_F$ from $\mathcal{T}$ as follows, where (T', χ') is a TD and φ, ψ are mappings from nodes to orderings. For every t of T, let $\chi'(t) := \chi(t)$, $\varphi_t := \varnothing$, $\psi_t := \varnothing$. For every two neighboring nodes t, t' of T with $t' \in \mathrm{chldr}(t)$ and $\mathrm{ord}(t) \times \mathrm{ord}(t') = \{(\alpha_1, \beta_1), \ldots, (\alpha_\ell, \beta_\ell)\}$, we add a sequence of fresh nodes $t_1, \ldots, t_\ell$ between t and t', such that for every $1 \leq i \leq \ell$ we define $\chi'(t_i) := \chi(t)$, $\varphi_{t_i} := \alpha_i$, and $\psi_{t_i} := \beta_i$. For every leaf node t of T with $\mathrm{ord}(t) = \{\alpha_1, \ldots, \alpha_\ell\}$, in T' we chain copy nodes $t_1, \ldots, t_\ell$ below t, where for every $1 \leq i \leq \ell$, $\chi'(t_i) := \chi(t)$, $\varphi_{t_i} := \alpha_i$, $\psi_{t_i} := \varnothing$.*

Observe that this definition provides the basis to analyze assignments in the form of different orderings, individually and one-by-one. Further, by comparing every pair of orderings of neighboring TD nodes, our reduction will later synchronize neighboring orderings and ensure compatibility.

Example 4. *Recall formula F and TD $\mathcal{T} = (T, \chi)$ from the previous example. Then, TD $\mathcal{T}$ can be turned into an ordering-augmented TD $\mathcal{T}' = (T', \chi', \varphi, \psi)$ according to Definition 1. Thereby we add a sequence (path) of child nodes to t_1; each of these nodes t_i handles one ordering φ_{t_i} over V_{t_1}. Similarly, this is carried out for node t_2. Between t_3 and t_1 we also add a path of fresh nodes, where each node covers a combination (pair) of orderings $\varphi_{t_3}, \psi_{t_1}$, which will be essentially used for synchronization. Analogously, this is done between t_3 and t_2.*

Involved Atoms. In order to guess among those $|\mathrm{ord}(t)|$ many orderings per node t of T, we characterize each ordering $\alpha \in \mathrm{ord}(t)$ by means of *atoms modeling ordering edges* of the form $e_{x,y}$ (and its negation $\hat{e}_{x,y}$) to indicate whether for every two different vertices $x, y \in V_t$, we have $x \prec_\alpha y$. Further, we require *atoms of the form* r_x for every $x \in V_t$, which stores whether x is reached or not. For V_t itself (which might span over several TD nodes) and $y \in V_t$ we require an additional (source) reachability atom of the form $r_{s_{V_t}}$ and edge atom $e_{s_{V_t}, y}$. Then, we also use (destination) reachability atom $r_{d_{V_t}}$ and edge atoms $e_{y, d_{V_t}}$ for every $y \in V_t$.

We will check, whether for every $x \in V_t$ of every node t of T, there is at *most one outgoing edge*, i.e., an atom $e_{x,y}$ contained in an answer set. To ensure this, we guide information of at most one outgoing edge along the tree decomposition, whereby for every node t of T and $x \in V_t$, we use an *auxiliary atom* o_t^x. This is also required for the source s_{V_t}, i.e., we also use *auxiliary atom* $o_t^{s_{V_t}}$ for every node t. Observe that this includes atoms $o_{t'}^{s_{V_t}}$ for nodes t', if $V_t = V_{t'}$.

In order to compare orderings, for every tree decomposition node t of T and element x contained in ordering α with $\alpha \in \{\varphi_t, \psi_t\}$, we use an additional *ordering query atom* q_t^x. If this query atom holds, the ordering α is at least fulfilled starting from the first atom contained in α up to the atom x. Intuitively, if q_t^y holds for the last atom y contained in α, we will ultimately be able to determine whether α holds.

These query atoms are supported by means of additional *testing points* p_t^x as well as *initial testing points* p_ϵ^x, which we use for every atom x contained in every $\alpha \in \{\varphi_t, \psi_t\}$ for every node t of T. The testing points allow us to check orderings with queries in every node. For every node t of T and atom $x \in V_t$, we refer by $\mathrm{prev}(x, t)$ to the *testing points of x preceding t*. Formally, $\mathrm{prev}(x, t) := \mathrm{chldr}(t)$ if $x \in V_{t'}$ for some $t' \in \mathrm{chldr}(t)$, otherwise $\{\epsilon\}$.

The Reduction. The reduction $\mathcal{R}$ takes a Boolean formula F and an ordering-augmented TD $\mathcal{T} = (T, \chi, \varphi, \psi)$. Before we commence with the description of $\mathcal{R}$, we require the following definition. For every node t of T, we let the *set E_t of ordering edges* be any arbitrary fixed subset of direct successors of φ_t. More precisely, for the first element $a \in \mathrm{dom}(\varphi_t)$ and the last element $b \in \mathrm{dom}(\varphi_t)$, i.e., a has no $\prec_{\varphi_t}$ predecessor and b has no $\prec_{\varphi_t}$ successor, E_t is the largest subset with $E_t \subseteq \{e_{x,y} \mid x \prec_{\varphi_t} y\} \cup \{e_{s_{V_t}, a}, e_{b, d_{V_t}}\}$ such that for any node t' of T with $t' \neq t$ we have $E_t \cap E_{t'} = \emptyset$.

Example 5. *Consider again formula F and ordering-augmented TD $\mathcal{T}' = (T', \chi', \varphi, \psi)$ from above. Observe that the definition of E_t is rather open, but it essentially requires that every ordering edge is encountered in exactly one node of T'. For example, the ordering $\varphi_{t^*} = \{v_1^1 \mapsto 0, v_1^2 \mapsto 1, v_1^2 \mapsto 2\}$ is handled in a node t^* below t_1. We could set $E_{t^*} = \{e_{s_{V_{t_1}}, v_1^1}, e_{v_1^1, v_1^2}, e_{v_1^2, v_1^3}, e_{v_1^3, d_{V_{t_1}}}\}$. Assume a different ordering $\varphi_{t'} = \{v_1^1 \mapsto 0, v_1^3 \mapsto 1, v_1^2 \mapsto 2\}$ over V_{t_1} in the child node t' of t^*. Then, considering E_{t^*}, i.e., avoiding overlapping edges, we could set $E_{t'} = \{e_{v_1^1, v_1^3}, e_{v_1^3, v_1^2}, e_{v_1^2, d_{V_{t_1}}}\}$. Similarly every remaining edge is covered uniquely in a node.*

Overall, the reduction $\mathcal{R}$ as given in Figure 2 consists of five blocks, where the first block of Formulas (1)–(7) concerns the *choice of orderings* (via edge atoms $e_{y,x}$) and enforces that every reachability atom r_x holds. Then, the second block of Formulas (8)–(10) takes care that for every node $y \in V_t$ of every node t of T, there is *at most one outgoing edge* of the form $e_{y,x}$ in an answer set. The third block of Formulas (11)–(13) ensures that *testing points* are maintained. This then allows us to properly *define ordering queries* in the forth block of Formulas (14)–(16). Finally, the last block of Formulas (17)–(19) takes care that chosen *orderings are compatible* and that every clause in F is satisfied.

Block 1: Choice of Orderings, Formulas (1)–(7). The first block concerns choosing orderings. Note that the disjunction of Formulas (3) is head-cycle-free and can be simply converted to normal rules by shifting (Ben-Eliyahu and Dechter 1994). Then, Formulas (1) set reachability of source vertices and Formulas (2) ensure reachability of all the vertices in V_t as well as the destination vertex for V_t, for every node t of T. Formulas (3) require to choose outgoing edges (at least

Block 1: Orderings & Reachability

$r_{s_{V_t}} \leftarrow$ for every t in T (1)

$\leftarrow \neg r_y$ for every t in $T, y \in V_t \cup \{d_{V_t}\}$ (2)

$e_{y,x} \vee \hat{e}_{y,x} \leftarrow r_y$ for every t in $T, e_{y,x} \in E_t$ (3)

$p^x_\epsilon \leftarrow e_{y,x}$ for every t in $T, e_{y,x} \in E_t, \{x, y\} \subseteq V_t$ (4)

$r_{d_{V_t}} \leftarrow e_{y,d_{V_t}}$ for every t in $T, e_{y,d_{V_t}} \in E_t$ (5)

$r_x \leftarrow p^x_{t'}$ for every t in $T, \varphi_t = \varnothing, t' \in \mathrm{chldr}(t), x \in \mathrm{dom}(\psi_{t'}) \setminus \mathrm{dom}(\varphi_{t'})$ (6)

$r_x \leftarrow p^x_{\mathrm{root}(T)}$ for every $x \in V_{\mathrm{root}(T)}$ (7)

Block 2: $\leq$1 Outgoing Edge

$o^y_t \leftarrow o^y_{t'}$ for every t in $T, t' \in \mathrm{chldr}(t), y \in (V_t \cap V_{t'}) \cup \{s_{V_t} \mid V_t = V_{t'}\}$ (8)

$o^y_t \leftarrow e_{y,x}$ for every t in $T, e_{y,x} \in E_t, y \in V_t$ (9)

$\leftarrow o^y_{t'}, e_{y,x}$ for every t in $T, e_{y,x} \in E_t, t' \in \mathrm{chldr}(t), y \in V_{t'} \cup \{s_{V_t} \mid V_t = V_{t'}\}$ (10)

Block 3: Testing Points

$p^x_t \leftarrow p^x_{t_1}, \ldots, p^x_{t_o}$ for every t in $T, x \in V_t, \varphi_t = \varnothing, \mathrm{prev}(x, t) = \{t_1, \ldots, t_o\}$, (11)

$p^x_t \leftarrow p^x_{t'}, \neg q^y_t$ for every t in $T, \alpha \in \{\varphi_t, \psi_t\}, \mathrm{prev}(x, t) = \{t'\}, \{x, y\} \subseteq \mathrm{dom}(\alpha), y \prec_\alpha x$ (12)

$p^x_t \leftarrow p^x_{t'}, \neg q^x_t$ for every t in $T, \alpha \in \{\varphi_t, \psi_t\}, \mathrm{prev}(x, t) = \{t'\}, x$ has no $\prec_\alpha$ successor (13)

Block 4: Ordering Queries

$q^x_t \leftarrow p^x_{t'}$ for every t in $T, \mathrm{prev}(x, t) = \{t'\}, \alpha \in \{\varphi_t, \psi_t\}, x \in \mathrm{dom}(\alpha), x$ has no $\prec_\alpha$ predecessor (14)

$q^x_t \leftarrow p^x_{t'}, q^y_t$ for every t in $T, \alpha \in \{\varphi_t, \psi_t\}, \mathrm{prev}(x, t) = \{t'\}, \{x, y\} \subseteq \mathrm{dom}(\alpha), y \prec_\alpha x$ (15)

$p^x_t \leftarrow q^x_t$ for every t in $T, \alpha \in \{\varphi_t, \psi_t\}, x \in \mathrm{dom}(\alpha)$ (16)

Block 5: Compatibility & SAT

$\leftarrow q^x_t, q^y_t$ for every t in $T, \alpha \in \{\varphi_t, \psi_t\}, x \in \mathrm{dom}(\alpha)$ has no $\prec_\alpha$ successor, (17)
$y \in \mathrm{dom}(\psi_t)$ has no $\prec_{\psi_t}$ successor, $\mathcal{I}_t(\varphi_t)$ and $\mathcal{I}_t(\psi_t)$ incompatible

$\leftarrow q^x_t$ for every t in $T, x \in \mathrm{dom}(\varphi_t)$ has no $\prec_{\varphi_t}$ successor, $\mathcal{I}_t(\varphi_t) \not\models F_t$ (18)

$\leftarrow q^x_t$ for every t in $T, x \in \mathrm{dom}(\varphi_t)$ has no $\prec_{\varphi_t}$ successor, $\mathcal{I}_t(\varphi_t)$ not defined (19)

Figure 2: The reduction $\mathcal{R}$ that takes a formula F and a corresponding ordering-augmented TD $\mathcal{T} = (T, \chi, \varphi, \psi)$ of $\mathcal{G}_F$.

one by Formulas (2)) from every reachable vertex y to some vertex x. This then yields initial testing points for x by Formulas (4). For the destination vertices of sets V_t, such testing points are not needed, so we immediately obtain reachability by Formulas (5). The connection and propagation between testing points will be achieved by Block 3. In the end, the last testing point for a vertex x yields reachability of x. This is ensured by Formulas (6), whenever x does not appear in an ordering for a successor node of t', or by Formulas (7), if x appears in an ordering of the root node.

Block 2: $\leq$ 1 Outgoing Edge, Formulas (8)–(10). This block ensures at most one outgoing edge per vertex y, where y can be also the source vertex s_{V_t} for a set V_t of ordering vertices. The information of whether y has decided an outgoing edge up to a node is propagated from a node t' to its parent node t by Formulas (8). Then, whenever in a node t an outgoing edge for y is chosen, o^y_t has to hold by Formulas (9). Finally, Formulas (10) prevent choosing outgoing edges for an atom y in a node t, if already chosen in a child node t'.

Block 3: Propagate Testing Points, Formulas (11)–(13). The third block concerns about propagation of testing points, if certain queries do not hold. For the case of the empty ordering, i.e., $\varphi_t = \varnothing$, in a node t, Formulas (11) directly propagate testing points for every atom $x \in V_t$ from the evidence of testing points for x in every child node of t (or from p^x_ϵ if $\mathrm{prev}(x, t) = \{\epsilon\}$). Further, whenever a certain ordering relation $y \prec_\alpha x$ for either $\alpha = \varphi_t$ or predecessor $\alpha = \psi_t$ does not hold, we still need to derive the corresponding testing point, see Formulas (12), as this testing point is required for further queries or for deriving reachability in the end, cf., Formulas (6), (7). This also holds for the very last element of α, see Formulas (13). The reason why we need to cover both orderings φ_t as well as ψ_t, is that neighboring orderings require compatibility, which will be discussed below Block 5.

Block 4: Define Ordering Queries, Formulas (14)–(16). This block focuses on deriving query atoms, which ensure that certain orderings hold. The first element of any ordering $\alpha \in \{\varphi_t, \psi_t\}$ is derived from the previous testing point, as given by Formulas (14). Then, whenever $y \prec_\alpha x$ is met, Formulas (15) enable to derive query atom q^x_t, which depends on the previous testing point for x as well as on q^y_t. This thereby ensures that the order $y \prec_\alpha x$ is indeed preserved, which is in contrast to Formulas (12) and (13) above. Finally, Formulas (16) immediately yield the corresponding testing point p^x_t in case query atom q^x_t holds.

Block 5: Compatibility of Orderings & Satisfiability, For-

mulas (17)–(19). The last block takes care of compatibility and satisfiability of every clause of the formula F by excluding orderings, whose corresponding assignments do not satisfy some clause. To this end, Formulas (17) excludes those cases of incompatible φ_t and ψ_t, i.e., to prevent inconsistencies, it is prohibited that query atoms q_t^x, q_t^y for the last element x of φ_t and the last element y of ψ_t hold. Most importantly, Formulas (18) ensure that the corresponding assignment of ordering φ_t satisfy clauses in F_t, in case the query atom q_t^x for the last element x of φ_t holds. Finally, Formulas (19) avoids corner cases, where unused orderings could be taken, which would enable bypassing satisfiability.

Example 6. *Recall formula F, ordering-augmented TD $\mathcal{T}'=(T',\chi',\varphi,\psi)$, as well as φ_{t^*} and E_{t^*} from Example 5. We briefly sketch the rules generated for node t^*.*

(1) $r_{s_{V_{t^*}}} \leftarrow$ (2) $\leftarrow r_{v_1^1}$; $\leftarrow r_{v_1^2}$; $\leftarrow r_{v_1^3}$; $\leftarrow \neg r_{V_{d_{V_{t^*}}}}$

(3) $e_{s_{V_{t^*}},v_1^1} \vee \hat{e}_{s_{V_{t^*}},v_1^1} \leftarrow r_{s_{V_{t^*}}}$; $e_{v_1^1,v_1^2} \vee \hat{e}_{v_1^1,v_1^2} \leftarrow r_{v_1^1}$;
$e_{v_1^2,v_1^3} \vee \hat{e}_{v_1^2,v_1^3} \leftarrow r_{v_1^2}$; $e_{v_1^3,d_{V_{t^*}}} \vee \hat{e}_{v_1^3,d_{V_{t^*}}} \leftarrow r_{v_1^3}$

(4) $p_\epsilon^{v_1^1} \leftarrow e_{s_{V_{t^*}},v_1^1}$; $p_\epsilon^{v_1^2} \leftarrow e_{v_1^1,v_1^2}$; $p_\epsilon^{v_1^3} \leftarrow e_{v_1^2,v_1^3}$

(5) $r_{v_1^3} \leftarrow e_{v_1^3,d_{V_{t^*}}}$

(8) $o_{t^*}^{v_1^1} \leftarrow o_{t'}^{v_1^1}$; $o_{t^*}^{v_1^2} \leftarrow o_{t'}^{v_1^2}$; $o_{t^*}^{v_1^3} \leftarrow o_{t'}^{v_1^3}$; $o_{t^*}^{d_{V_{t^*}}} \leftarrow o_{t'}^{d_{V_{t^*}}}$

(9) $o_{t^*}^{s_{V_{t^*}}} \leftarrow e_{s_{V_{t^*}},v_1^1}$; $o_{t^*}^{v_1^1} \leftarrow e_{v_1^1,v_1^2}$; $o_{t^*}^{v_1^2} \leftarrow e_{v_1^2,v_1^3}$;
$o_{t^*}^{v_1^3} \leftarrow e_{v_1^3,d_{V_{t^*}}}$

(10) $\leftarrow o_{t'}^{s_{V_{t^*}}}, e_{s_{V_{t^*}},v_1^1}$; $\leftarrow o_{t'}^{v_1^1}, e_{v_1^1,v_1^2}$; $\leftarrow o_{t'}^{v_1^2}, e_{v_1^2,v_1^3}$;
$\leftarrow o_{t'}^{v_1^3}, e_{v_1^3,d_{V_{t^*}}}$

(11) $p_{t^*}^{v_1^1} \leftarrow p_{t'}^{v_1^1}$; $p_{t^*}^{v_1^2} \leftarrow p_{t'}^{v_1^2}$; $p_{t^*}^{v_1^3} \leftarrow p_{t'}^{v_1^3}$

(12) $p_{t^*}^{v_1^2} \leftarrow p_{t'}^{v_1^2}, \neg q_{t^*}^{v_1^1}$; $p_{t^*}^{v_1^3} \leftarrow p_{t'}^{v_1^3}, \neg q_{t^*}^{v_1^2}$

(13) $p_{t^*}^{v_1^3} \leftarrow p_{t'}^{v_1^3}, \neg q_{t^*}^{v_1^3}$

(14) $q_{t^*}^{v_1^1} \leftarrow p_{t'}^{v_1^1}$ (15) $q_{t^*}^{v_1^2} \leftarrow p_{t'}^{v_1^2}, q_{t^*}^{v_1^1}$; $q_{t^*}^{v_1^3} \leftarrow p_{t'}^{v_1^3}, q_{t^*}^{v_1^2}$

(16) $p_{t^*}^{v_1^1} \leftarrow q_{t^*}^{v_1^1}$; $p_{t^*}^{v_1^2} \leftarrow q_{t^*}^{v_1^2}$; $p_{t^*}^{v_1^3} \leftarrow q_{t^*}^{v_1^3}$

Note that if $\mathcal{I}_{t^*}(\varphi_{t^*}) \not\models F_{t^*}$ or $\mathcal{I}_{t^*}(\varphi_{t^*})$ is undefined (unused ordering), Formulas (18) and (19) generate $\leftarrow q_{t^*}^{v_1^3}$.

Properties and Consequences of the Reduction

Next, we show that the reduction indeed utilizes the structural parameter treewidth, i.e., the treewidth is decreased.

Theorem 1 (Treewidth-Awareness)**.** *The reduction from a Boolean formula F and an ordering-augmented TD $\mathcal{T}=(T, \chi, \varphi, \psi)$ of $\mathcal{G}_F$ to normal program Π consisting of Formulas (1) to (19) slightly decreases treewidth. Precisely, if k is the width of $\mathcal{T}$, the treewidth of $\mathcal{G}_\Pi$ is in $\mathcal{O}(\frac{k}{\log(k)})$.*

Proof (Sketch). We construct a TD $\mathcal{T}' = (T, \chi')$ of $\mathcal{G}_\Pi$ to show that the width of $\mathcal{T}'$ increases only slightly (compared to k). To this end, let t be a node of T with $\mathrm{chldr}(t) = \langle t_1, \ldots, t_\ell \rangle$ and let $\hat{t}$ be the parent of t (if exists). We define (i) $R(t) := \{r_x \mid x \in V_t\} \cup \{r_x \mid x \in V_{t'}, t' \in \mathrm{chldr}(t)\} \cup \{r_{s_{V_t}}, r_{d_{V_t}}\}$, (ii) $E(t) := \{e_{x,y}, \hat{e}_{x,y} \mid e_{x,y} \in E_t\}$, (iii) $P(t) := \{q_t^x, p_t^x, p_{t'}^x, p_\epsilon^x \mid x \in \mathrm{dom}(\varphi_t) \cup \mathrm{dom}(\psi_t), t' \in \mathrm{prev}(x,t)\}$, and (iv) $O(t) := \{o_t^x, o_t^{s_{V_t}} \mid x \in V_t\} \cup \{o_{t'}^x, o_{t'}^{s_{V_{t'}}} \mid t' \in \mathrm{chldr}(t), x \in V_{t'} \cap V_t\}$. Then, we let $\chi'(t) := R(t) \cup E(t) \cup P(t) \cup O(t)$. Observe that $\mathcal{T}'$ is a TD of $\mathcal{G}_\Pi$ and by construction $|\chi'(t)|$ is in $\mathcal{O}(|V_t|)$. By definition of V_t, $|V_t| \leq k'$, where $k'! \geq 2^k$. Then, since $k'!$ is in $2^{\Omega(k' \cdot \log(k'))}$ (see Observation 1), we have that $2^{\Omega(k' \cdot \log(k'))}$ is at least 2^k and therefore $k' = \mathcal{O}(\frac{k}{\log(k')}) = \mathcal{O}(\frac{k}{\log(k) - \log(\log(k'))}) = \mathcal{O}(\frac{k}{\log(k)})$. ☐

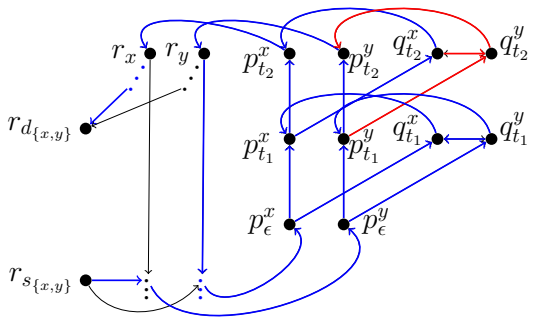

Figure 3: Positive dependency graph over Blocks 1,3, and 4 of reduction $\mathcal{R}$ for two TD nodes t_1, t_2 over ordering vertices $V_{t_1}=V_{t_2}=\{x,y\}$. These blocks potentially cause cycles (larger SCCs). The parts highlighted in blue are the atoms in their order of derivation, assuming that y precedes x, i.e., $y \prec_{\varphi_{t_1}} x$. Edges highlighted in red cannot be taken, i.e., q_2^y cannot be derived, since this implies $x \prec_{\varphi_{t_2}} y$, contradicting $y \prec_{\varphi_{t_1}} x$; causing a cycle of unproven atoms involving r_y.

Interestingly, our reduction cannot be significantly improved, making further treewidth decreases unlikely.

Theorem 2 (Treewidth Decrease is Optimal)**.** *Assume a reduction from a formula F to a normal program Π, running in time $2^{o(k)} \cdot \mathrm{poly}(|\mathrm{at}(F)|)$, where $k = tw(\mathcal{G}_F)$. Then, unless ETH fails, the treewidth of $\mathcal{G}_\Pi$ cannot be in $o(\frac{k}{\log(k)})$.*

Proof. Assume towards a contradiction that such a reduction, call it $\mathcal{R}^*$, exists. Then, we apply this reduction on any F, resulting in program $\Pi = \mathcal{R}^*(F)$. Then, we know that Π can be decided (Hecher 2022) in time $2^{\mathcal{O}(k' \cdot \log(k'))} \cdot \mathrm{poly}(|\mathrm{at}(F)|)$, where k' is in $o(\frac{k}{\log(k)})$. As a result, we have that Π and therefore F can be decided in time $2^{o(\frac{k}{\log(k)} \cdot \log(k))} \cdot \mathrm{poly}(|\mathrm{at}(F)|)$, which is in $2^{o(k)} \cdot \mathrm{poly}(|\mathrm{at}(F)|)$, contradicting the ETH. ☐

Correctness establishes that the reduction $\mathcal{R}$ encoded by Formulas (1) to (19) indeed characterizes the satisfying assignments of a Boolean formula.

Theorem 3 (Correctness)**.** *The reduction from a Boolean formula F and a TD $\mathcal{T} = (T, \chi)$ of $\mathcal{G}_\Pi$ to a logic program Π, consisting of Formulas (1) to (19), is correct. Concretely, for every model of F, there is an answer set of Π. Vice versa, for every answer set of Π, there is a unique model of F.*

Example 7. *Figure 3 sketches the dependency graph over the rules of Blocks 1,3, and 4 on two simple TD nodes; showing how incompatible queries would cause cyclic dependencies that are unproven, which can therefore not occur.*

4 Further Insights Into Hardness of ASP

In this section, we provide deeper insights into the characterization of hardness for normal logic programs. First, we discuss consequences on the length of the largest SCC.

Are Unbounded Cycles (SCCs) Vital for Hardness?

By the construction of the reduction in Section 3 and the observation that $\mathcal{R}$ causes cycles (SCCs) in the program's dependency graph of size $2^{\mathcal{O}(k \cdot \log(k))}$, we obtain the following new hardness and precise lower bound result for deciding the consistency of normal programs.

Corollary 1 (LB Largest SCC)**.** *Let Π be a normal logic program, where the treewidth of $\mathcal{G}_\Pi$ is k such that the largest SCC size of D_Π is in $2^{\mathcal{O}(k \cdot \log(k))}$. Under ETH, the consistency of Π can not be decided in time $2^{o(k \cdot \log(k))} \cdot o(|\mathrm{at}(\Pi)|)$.*

Corollary 1 gives insights into the SCC size required for hardness, which is in contrast to known lower bounds, which could not bound the cycle length or SCC size in the treewidth (Hecher 2022). Interestingly, this corollary is in line with the corresponding upper bound of Proposition 1 (for $\iota = k$). We do not expect that Corollary 1 can be significantly strengthened, but we show below how for ι-tight programs the SCC size can be decreased to $2^{o(k \cdot \log(\iota))}$.

This also provides the corresponding lower bound for projected answer set counting, which was left open (Fichte and Hecher 2019). Our reduction allows us to close the gap to the upper bound, showing that it is expected for the problem to be harder than plain counting on disjunctive programs.

Theorem 4 (LB Projected Counting)**.** *Let Π be a normal logic program, $P \subseteq \mathrm{at}(\Pi)$ be atoms, and k be the treewidth of $\mathcal{G}_\Pi$, such that the largest SCC size of D_Π is in $2^{\mathcal{O}(k \cdot \log(k))}$. Then, under ETH, the cardinality $|\{M \cap P \mid M \models \Pi\}|$ cannot be computed in time $2^{2^{o(k \cdot \log(k))}} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$.*

An ETH-Tight Lower Bound for ι-Tight Programs

Recall the fragment of ι-tight programs, which is motivated by the idea of providing almost tight programs that are simpler to solve than normal programs. Every formula F can be compiled into a (1-)tight program, cf., Example 1.

As a result, in the following, we generalize our reduction $\mathcal{R}$ of the previous section from $\iota = k$ to the case $\iota \geq 2$, resulting in $\mathcal{R}'$. Intuitively, this reduction $\mathcal{R}'$ allows us to decrease the treewidth k, but not necessarily to the maximum of $\mathcal{O}(\frac{k}{\log(k)})$ of normal programs. Instead, ι provides a precise handle on the tightness, thereby decreasing treewidth to $\mathcal{O}(\frac{k}{\log(\iota)})$.

Adapted Reduction. We adapt the construction of reduction $\mathcal{R}$ and obtain $\mathcal{R}'$. To this end, we take an instance F of SAT, i.e., a Boolean formula, and an ordering-augmented TD $\mathcal{T} = (T, \chi, \varphi, \psi)$ of $\mathcal{G}_F$ of width k. Then, we simulate for each node of T, the up to 2^k many assignments via $(\iota!)^{\frac{k'}{\iota}}$ many orderings, where ι is any fixed $2 \leq \iota \leq k'$. So we decrease the treewidth from k to k' such that $(\iota!)^{\frac{k'}{\iota}} \geq 2^k$, where the special case of the previous section corresponds to $\iota = k'$. Consequently, since there are still up to k' many elements per bag, but we only have ι many positions, we require $\frac{k'}{\iota}$ many SCCs per bag. As a result, the orderings $\mathrm{ord}(t)$ for a node t are not total. So, instead of one source and destination vertex for ordering set V_t, we require up to $\frac{k'}{\iota}$ reachability atoms of the form $r_{s^j_{V_t}}, r_{d^j_{V_t}}$ for every $1 \leq j \leq \frac{k'}{\iota}$. This requires minor adaptions in the definition of E_t, Formulas (1), (2), (5), as well as (8) and (10), as sketched in an extended version.

One can show a generalization of Observation 1, where $\iota < k$.

Observation 2. *Let $2 \leq \iota \leq k$. Then, $(\iota!)^{\frac{k}{\iota}}$ is in $2^{\Omega(k \cdot \log(\iota))}$.*

By Observation 2, $(\iota!)^{\frac{k'}{\iota}}$ is in $2^{\Omega(k' \cdot \log(\iota))}$, so we have that $2^{\Omega(k' \cdot \log(\iota))}$ is at least 2^k and therefore $k' = \mathcal{O}(\frac{k}{\log(\iota)})$. As a result, $\mathcal{R}'$ slightly reduces treewidth to $\mathcal{O}(\frac{k}{\log(\iota)})$. Consequently, for the result as given in Proposition 1 it is unexpected that it can be significantly improved (under ETH). More precisely, we obtain the following lower bound result.

Theorem 5 (LB ι-Tightness)**.** *Let Π be a ι-tight logic program, where the treewidth of $\mathcal{G}_\Pi$ is k such that the largest SCC size of D_Π is in $2^{\mathcal{O}(k \cdot \log(\iota))}$. Then, under ETH, the consistency of Π cannot be decided in $2^{o(k \cdot \log(\iota))} \cdot o(|\mathrm{at}(\Pi)|)$.*

5 Discussion and Conclusion

The complexity of ASP has already been studied in different facets and flavors. Recently, it has been shown that under the exponential time hypothesis (ETH), the evaluation of normal logic programs is expected to be *slightly harder* for the structural parameter treewidth, than deciding satisfiability (SAT) of Boolean formulas. However, the hardness proof relies on *large cycles (SCCs)*, unbounded in the treewidth. Further, compared to standard reductions, see Example 1, utilizing the "hardness" of normal ASP remained unclear.

In this paper, we address both shortcomings. The idea is to reduce from SAT to normal ASP, thereby actively decreasing structural dependency in the form of treewidth. We design such a reduction that *reduces treewidth* from k to $\frac{k}{\log(k)}$. We show that under ETH, this decrease cannot be significantly improved. Even further, with the help of the reduction, the existing hardness result for normal programs and treewidth can be improved: The constructed cycles (SCCs) are not required to be unbounded in the treewidth; indeed, hardness is preserved in case of a *single-exponential bound in the treewidth*. Then, we further improve this bound for the class of ι-tight programs, which allows us to *close the gap* to the known upper bound, which our results render ETH-tight.

Finally, we apply our reduction for establishing further ETH-tight lower bounds on normal logic programs. We hope that the reduction of this work enables further consequences and insights on hardness for logic programs. In the light of a known result (Atserias, Fichte, and Thurley 2011) on the correspondence of treewidth and resolution width applied in SAT solving, this might pave the way towards such insights for ASP. Currently, we are working on the comparison of different reductions from SAT to ASP and how they perform in practice. Given the unsuccessful application of directed measures for ASP (Bliem, Ordyniak, and Woltran 2016), structural parameters between treewidth and directed variants could lead to new insights.

Acknowledgments

This work has been supported by the Austrian Science Fund (FWF), Grants J 4656 and P32830, the Society for Research Funding in Lower Austria (GFF NÖ) grant ExzF-0004, as well as the Vienna Science and Technology Fund, Grant WWTF ICT19-065.

References

Abels, D.; Jordi, J.; Ostrowski, M.; Schaub, T.; Toletti, A.; and Wanko, P. 2019. Train Scheduling with Hybrid ASP. In *LPNMR*, volume 11481 of *LNCS*, 3–17. Springer.

Alviano, M.; Amendola, G.; Dodaro, C.; Leone, N.; Maratea, M.; and Ricca, F. 2019. Evaluation of Disjunctive Programs in WASP. In *LPNMR'19*, volume 11481 of *LNCS*, 241–255. Springer.

Alviano, M.; and Dodaro, C. 2016. Completion of Disjunctive Logic Programs. In *IJCAI'16*, 886–892. IJCAI/AAAI Press.

Atserias, A.; Fichte, J. K.; and Thurley, M. 2011. Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution. *J. Artif. Intell. Res.*, 40: 353–373.

Balduccini, M.; Gelfond, M.; and Nogueira, M. 2006. Answer set based design of knowledge systems. *Ann. Math. Artif. Intell.*, 47(1-2): 183–219.

Ben-Eliyahu, R.; and Dechter, R. 1994. Propositional Semantics for Disjunctive Logic Programs. *Ann. Math. Artif. Intell.*, 12(1): 53–87.

Bichler, M.; Morak, M.; and Woltran, S. 2020. selp: A Single-Shot Epistemic Logic Program Solver. *Theory Pract. Log. Program.*, 20(4): 435–455.

Bidoít, N.; and Froidevaux, C. 1991. Negation by default and unstratifiable logic programs. *Theo. Comput. Science*, 78(1): 85–112.

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds. 2009. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artif. Intellig. and Applications*. IOS Press. ISBN 978-1-58603-929-5.

Bliem, B.; Morak, M.; Moldovan, M.; and Woltran, S. 2020. The Impact of Treewidth on Grounding and Solving of Answer Set Programs. *J. Artif. Intell. Res.*, 67: 35–80.

Bliem, B.; Ordyniak, S.; and Woltran, S. 2016. Clique-Width and Directed Width Measures for Answer-Set Programming. In *ECAI'16*, volume 285 of *FAIA*, 1105–1113. IOS Press.

Bomanson, J. 2017. lp2normal - A Normalization Tool for Extended Logic Programs. In *LPNMR'17*, volume 10377 of *LNCS*, 222–228. Springer.

Bomanson, J.; and Janhunen, T. 2013. Normalizing Cardinality Rules Using Merging and Sorting Constructions. In *LPNMR'13*, volume 8148 of *LNCS*, 187–199. Springer.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Comm. of the ACM*, 54(12): 92–103.

Clark, K. L. 1977. Negation as Failure. In *Logic and Data Bases*, Advances in Data Base Theory, 293–322. Plemum Press.

Cygan, M.; Fomin, F. V.; Kowalik, Ł.; Lokshtanov, D.; Dániel Marx, M. P.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer. ISBN 978-3-319-21274-6.

Diestel, R. 2012. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer. ISBN 978-3-642-14278-9.

Downey, R. G.; and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer. ISBN 978-1-4471-5558-4.

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.

Fandinno, J.; and Hecher, M. 2021. Treewidth-Aware Complexity in ASP: Not all Positive Cycles are Equally Hard. In *AAAI'21*, 6312–6320. AAAI Press.

Fichte, J. K.; and Hecher, M. 2019. Treewidth and Counting Projected Answer Sets. In *LPNMR'19*, volume 11481 of *LNCS*, 105–119. Springer.

Fichte, J. K.; Hecher, M.; and Pfandler, A. 2020. Lower Bounds for QBFs of Bounded Treewidth. In *LICS'20*, 410–424. Assoc. Comput. Mach.

Flum, J.; and Grohe, M. 2006. *Parameterized Complexity Theory*, volume XIV of *Theo. Comput. Science*. Springer. ISBN 978-3-540-29952-3.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2009. Solution Enumeration for Projected Boolean Search Problems. In *CPAIOR'09*, volume 5547 of *LNCS*, 71–86. Springer. ISBN 978-3-642-01929-6.

Gelfond, M.; and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.*, 9(3/4): 365–386.

Gottlob, G.; Scarcello, F.; and Sideri, M. 2002. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artif. Intell.*, 138(1-2): 55–86.

Guziolowski, C.; Videla, S.; Eduati, F.; Thiele, S.; Cokelaer, T.; Siegel, A.; and Saez-Rodriguez, J. 2013. Exhaustively characterizing feasible logic models of a signaling network using Answer Set Programming. *Bioinformatics*, 29(18): 2320–2326. Erratum see Bioinformatics 30, 13, 1942.

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

Impagliazzo, R.; Paturi, R.; and Zane, F. 2001. Which Problems Have Strongly Exponential Complexity? *J. of Computer and System Sciences*, 63(4): 512–530.

Jakl, M.; Pichler, R.; and Woltran, S. 2009. Answer-Set Programming with Bounded Treewidth. In *IJCAI'09*, volume 2, 816–822.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *J. of Applied Non-Classical Logics*, 16(1-2): 35–86.

Janhunen, T.; and Niemelä, I. 2016. The Answer Set Programming Paradigm. *AI Magazine*, 37(3): 13–24.

Kleine Büning, H.; and Lettman, T. 1999. *Propositional logic: deduction and algorithms*. Cambridge University Press. ISBN 978-0521630177.

Kloks, T. 1994. *Treewidth. Computations and Approximations*, volume 842 of *LNCS*. Springer. ISBN 3-540-58356-4.

Lackner, M.; and Pfandler, A. 2012. Fixed-Parameter Algorithms for Finding Minimal Models. In *KR'12*. 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, J. 2003. On tight logic programs and yet another translation from normal logic programs to propositional logic. In *IJCAI'03*, 853–858. Morgan Kaufmann.

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

Lokshtanov, D.; Marx, D.; and Saurabh, S. 2011. Slightly Superexponential Parameterized Problems. In *SODA'11*, 760–776. SIAM.

Marek, W.; and Truszczyński, M. 1991. Autoepistemic logic. *J. of the ACM*, 38(3): 588–619.

Marx, D.; and Mitsou, V. 2016. Double-Exponential and Triple-Exponential Bounds for Choosability Problems Parameterized by Treewidth. In *ICALP'16*, volume 55 of *LIPIcs*, 28:1–28:15. Dagstuhl Publishing. ISBN 978-3-95977-013-2.

Niedermeier, R. 2006. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press. ISBN 978-0-19-856607-6.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable model semantics of weight constraint rules. In *LPNMR'99*, volume 1730 of *LNCS*, 317–331. Springer. ISBN 3-540-66749-0.

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. An A-Prolog Decision Support System for the Space Shuttle. In *PADL'01*, volume 1990 of *LNCS*, 169–183. Springer. ISBN 978-3-540-45241-6.

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).

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

Schaub, T.; and Woltran, S. 2018. Special Issue on Answer Set Programming. *KI*, 32(2-3): 101–103.

Truszczynski, M. 2011. Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs. *Theory Pract. Log. Program.*, 11(6): 881–904.