Treewidth-Aware Complexity in ASP: Not all Positive Cycles are Equally Hard*

Jorge Fandinno[1,2] **and Markus Hecher**[2,3]
[1]University of Nebraska Omaha, USA
[2]University of Potsdam, Germany
[3]TU Wien, Austria
jfandinno@unomaha.edu, mhecher@gmail.com

Abstract

It is well-known that deciding consistency for normal answer set programs (ASP) is NP-complete, thus, as hard as the satisfaction problem for propositional logic (SAT). The *exponential time hypothesis* (ETH) implies that the best algorithms to solve these problems take exponential time in the worst case. However, accounting for the treewidth, the consistency problem for ASP is slightly harder than SAT: while SAT can be solved by an algorithm that runs in exponential time in the treewidth k, normal ASP requires exponential time in $k \cdot \log(k)$. This extra cost is due to checking that there are no self-supported true atoms because of positive cycles in the program. In this paper, we refine this recent result and show that consistency for ASP can be decided in exponential time in $k \cdot \log(\iota)$ where ι is a novel measure, bounded by both treewidth k and the size ℓ of the largest strongly-connected component of the positive dependency graph of the program. We provide a treewidth-aware reduction from ASP to SAT that adheres to the above limit.

1 Introduction

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011; Gebser et al. 2012) is a problem modeling and solving paradigm well-known in the area of knowledge representation and reasoning that is experiencing an increasing number of successful applications (Balduccini, Gelfond, and Nogueira 2006; Nogueira et al. 2001; Guziolowski et al. 2013). The flexibility of ASP comes with a high computational complexity cost: its *consistency problem*, that is, deciding the existence of a solution (*answer set*) for a given logic program is Σ_2^P-complete (Eiter and Gottlob 1995), in general. Fragments with lower complexity are also known. For instance, the consistency problem for *normal*, *head-cycle-free (HCF)*, or *tight* ASP is NP-complete. Even for solving this class of programs, the best known algorithms require exponential time with respect to the size of the program. Still, existing solvers (Gebser et al. 2012; Alviano et al. 2017) are able to find solutions for many interesting problems in reasonable time. A way to shed light into this discrepancy is by means of *parameterized complexity* (Cygan et al. 2015), which conducts more fine-grained complexity analysis in terms of parameters of a problem. For ASP, several results were achieved in this direction (Gottlob, Scarcello, and Sideri 2002; Lonc and Truszczynski 2003; Lin and Zhao 2004; Fichte and Szeider 2015), some insights involve even combinations (Lackner and Pfandler 2012; Fichte, Kronegger, and Woltran 2019) of parameters. More recent studies focus on the influence of the parameter *treewidth* for solving ASP (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017; Fichte and Hecher 2019; Bichler, Morak, and Woltran 2018; Bliem et al. 2020). These works utilize treewidth to solve, e.g., the consistency problem, in polynomial time in the program size, while being superpolynomial only in the treewidth. Recently, it was shown that for normal ASP deciding consistency is expected to be slightly superexponential in the treewidth (Hecher 2020). More concretely, a lower bound was established under the *Exponential Time Hypothesis (ETH)* (Impagliazzo, Paturi, and Zane 2001), saying that consistency for any normal logic program of treewidth k cannot be decided in time $2^{o(k \cdot \log(k))} \cdot \mathrm{poly}(n)$, where n is the *number of variables (atoms) of the program*. This result matches the known upper bound (Fichte and Hecher 2019) and renders the consistency of normal ASP slightly harder than the satisfiability (SAT) of a propositional formula, which under the ETH cannot be decided in time $2^{o(k)} \cdot \mathrm{poly}(n)$.

Contributions. We address this result and consider besides treewidth, the size ℓ of the largest strongly-connected component (SCC) of the *positive dependency graph* as parameter.

1. First, we present a treewidth-aware reduction from head-cycle-free ASP to tight ASP. Our reduction takes any HCF program Π and creates a tight program, whose treewidth is at most $\mathcal{O}(k \cdot \log(\ell))$, where k is the treewidth of Π and ℓ is the size of the largest SCC of the dependency graph of Π. In general, the treewidth of the resulting tight program cannot be in $o(k \cdot \log(k))$, unless ETH fails. Our reduction forms a major improvement for the case $\ell \ll k$. Besides, this reduction bijectively preserves answer sets and can be used for counting and enumerating answer sets.
2. Then, we improve known results (Fichte and Hecher 2019; Hecher 2020) for the consistency of head-cycle-free ASP. We define a novel measure ι for the "level" tightness of pro-

*This work has been supported by the Austrian Science Fund (FWF), Grants Y698 and P32830, and the Vienna Science and Technology Fund, Grant WWTF ICT19-065.

grams that is bounded by *both* treewidth k and the largest SCC size ℓ such that ι=1 for tight programs only. Interestingly, there are programs with large cycles (large values of ℓ), where we have ι=2. We present a treewidth-aware reduction, capable of "increasing" tightness at the cost of higher treewidth, which increases up to $\mathcal{O}(k \cdot \log(\iota))$. Further, we establish an algorithm for consistency, running in time $2^{\mathcal{O}(k \cdot \log(\iota))} \cdot \mathrm{poly}(n)$.

3. Finally, we show a treewidth-aware reduction that takes any tight logic program Π and creates a propositional formula, whose treewidth is linear in the treewidth of the program. This reduction cannot be significantly improved under ETH. Our result also establishes that for deciding consistency of tight logic programs of bounded treewidth k, one indeed obtains the same runtime as for SAT, namely $2^{\mathcal{O}(k)} \cdot n$, which is ETH-tight.

Related Work. While the largest SCC size has already been considered (Janhunen 2006), it has not been studied in combination with treewidth. Also programs, where the number of even and/or odd cycles is bounded, have been analyzed (Lin and Zhao 2004), which is orthogonal to the size of the largest cycle or largest SCC size ℓ. Indeed, in the worst-case, each component might have an exponential number of cycles in ℓ. Further, the literature distinguishes the so-called feedback width (Gottlob, Scarcello, and Sideri 2002), depending on the atoms required to break large SCCs (positive cycles). There are related measures, called smallest backdoor size, where the removal of a set of atoms from the program results in normal or acyclic programs (Fichte and Szeider 2015).

2 Background

We assume familiarity with graph terminology. Given a directed graph $G = (V, E)$. 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. An SCC C is called *non-trivial*, if $|C| > 1$. A *cycle* over some vertex v of G is a directed path from v to v.

Answer Set Programming (ASP). We assume familiarity with propositional satisfiability (SAT) and follow standard definitions of ASP (Brewka, Eiter, and Truszczyński 2011). Let m, n, o be non-negative integers such that $m \leq n \leq o$, and let $a_1, \ldots, a_o$ be distinct propositional atoms. Moreover, we refer by *literal* to an atom or the negation thereof. A *(logic) program* Π is a set of *rules* of the form $a_1 \vee \cdots \vee a_m \leftarrow a_{m+1}, \ldots, a_n, \neg a_{n+1}, \ldots, \neg a_o$. For a rule r, we let $H_r := \{a_1, \ldots, a_m\}$, $B_r^+ := \{a_{m+1}, \ldots, a_n\}$, and $B_r^- := \{a_{n+1}, \ldots, a_o\}$. 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 $\mathrm{at}(\Pi) := \bigcup_{r \in \Pi} \mathrm{at}(r)$. For a set $X \subseteq \mathrm{at}(\Pi)$ of atoms, we let $\overline{X} := \{\neg x \mid x \in X\}$. A program Π is *normal*, if $|H_r| \leq 1$ for every $r \in \Pi$. The *(positive) dependency digraph* D_Π of Π is the directed graph defined on the set of atoms from $\bigcup_{r \in \Pi} H_r \cup B_r^+$, where there is a directed edge from vertex a to vertex b iff there is a rule $r \in \Pi$ with $a \in B_r^+$ and $b \in H_r$. A head-cycle of D_Π is a cycle containing two distinct atoms $a, b \in H_r$ for some rule $r \in \Pi$. A program Π is *head-cycle-free (HCF)* if D_Π contains no head-cycle (Ben-Eliyahu and Dechter 1994) and Π is called *tight* (Lin and Zhao 2003) if D_Π contains no cycle, i.e., Π has no "*positive cycle*". The class of tight, normal, and HCF programs is referred to by *tight, normal, and HCF ASP*, respectively.

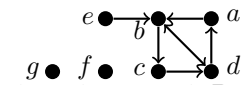

Figure 1: Positive dependency graph D_Π of Π of Example 1.

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. For brevity, we view propositional formulas as sets of clauses and use the notion of interpretations, models, and satisfiability analogously. Given a set $A \subseteq \mathrm{at}(\Pi)$ of atoms, a function $\sigma : A \to \{0, \ldots, |A| - 1\}$ is called *level mapping* over A. Given an interpretation I of a normal program Π and a level mapping σ over I, an atom $a \in I$ is *proven* if there is a rule $r \in \Pi$ *proving* a *with* σ, 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) $\sigma(b) < \sigma(a)$ 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 with σ. This characterization (Lin and Zhao 2003) vacuously extends to HCF programs and allows for further simplification when considering SCCs of D_Π (Janhunen 2006). To this end, we denote for each atom $a \in \mathrm{at}(\Pi)$ the *SCC of atom* a in D_Π by $\mathrm{scc}(a)$. Then, Condition (iii) above can be relaxed to $\sigma(b) < \sigma(a)$ for every $b \in B_r^+ \cap \mathrm{scc}(a)$.

The problem of deciding whether an ASP program has an answer set is called *consistency*, which is NP-complete for normal, HCF, and tight programs (Marek and Truszczyński 1991; Ben-Eliyahu and Dechter 1994; Lin and Zhao 2003). In general, semantics are defined via the GL-reduct (Gelfond and Lifschitz 1991), where for arbitrary programs the complexity increases to Σ_2^P-complete (Eiter and Gottlob 1995).

Example 1. *Consider program* $\Pi := \{r_1, \ldots, r_7\}$*, where* $r_1 := a \leftarrow d$, $r_2 := b \leftarrow a$, $r_3 := b \leftarrow d$, $r_4 := b \leftarrow e, \neg f$, $r_5 := c \leftarrow b$, $r_6 := d \leftarrow b, c$, *and* $r_7 := e \vee f \vee g \leftarrow$. *Observe that* Π *is head-cycle-free. Figure 1 shows the positive dependency graph* D_Π *consisting of SCCs* $\mathrm{scc}(e), \mathrm{scc}(f), \mathrm{scc}(g)$, *and* $\mathrm{scc}(a) = \mathrm{scc}(b) = \mathrm{scc}(c) = \mathrm{scc}(d)$. *Then,* $I := \{a, b, c, d, e\}$ *is an answer set of* Π*, since* I *is a model of* Π *and with level mapping* $\sigma := \{e \mapsto 0, b \mapsto 0, c \mapsto 1, d \mapsto 2, a \mapsto 3\}$*, we prove* e *by rule* r_7*,* b *by rule* r_4 *since* $e \notin B_{r_4}^+ \cap \mathrm{scc}(b)$*,* c *by* r_5*,* d *by* r_6*, and* a *by* r_1*. Further answer sets are* $\{f\}$ *and* $\{g\}$.

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) "connectedness": 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 $\mathrm{chld}(t)$ the *set of child nodes of* t in T. Any TD can be turned into a TD with constantly many child nodes (Kloks 1994) per node in linear time. 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. TDs can be approximated efficiently (Bodlaender et al. 2016).

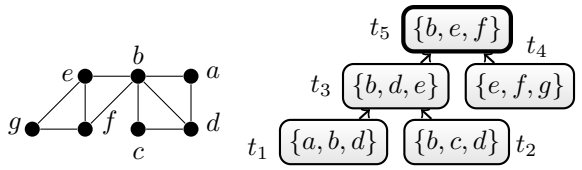

Figure 2: Graph G (left) and a TD $\mathcal{T}$ of G (right).

Example 2. *Figure 2 illustrates a graph G and a TD $\mathcal{T}$ of G of width 2, which is also the treewidth of G, since vertices b,c,d are completely connected with each other (Kloks 1994).*

In order to use TDs, we need dedicated graph representations. The *primal graph* G_Π of a program Π (Jakl, Pichler, and Woltran 2009) has the atoms of Π as vertices and an edge $\{a, b\}$ if there exists a rule $r \in \Pi$ and $a, b \in \mathrm{at}(r)$, and the *treewidth of* Π equals $tw(G_\Pi)$. Similarly, G_F denotes the primal graph of a formula F and the *treewidth* of F equals $tw(G_F)$. Let $\mathcal{T} = (T, \chi)$ be a TD of primal graph G_Π, and let t be a node of T. Then, the *bag program* Π_t contains rules covered by $\chi(t)$, i.e., $\Pi_t := \{r \mid r \in \Pi, \mathrm{at}(r) \subseteq \chi(t)\}$.

Example 3. *Recall program Π from Example 1 and observe that graph G of Figure 2 is the primal graph G_Π of Π. Further, we have $\Pi_{t_1} = \{r_1, r_2, r_3\}$, $\Pi_{t_2} = \{r_3, r_5, r_6\}$, $\Pi_{t_3} = \emptyset$, $\Pi_{t_4} = \{r_7\}$, and $\Pi_{t_5} = \{r_4\}$.*

3 Bounding Treewidth and Positive Cycles

Recently, it was shown that under reasonable assumptions, namely the exponential time hypothesis (ETH), the consistency of normal logic programs is slightly superexponential and one cannot significantly improve in the worst case.

Proposition 1 (Lower Bound (Hecher 2020))**.** *Given a normal or HCF logic program Π, where k is the treewidth of the primal graph of Π. Then, under ETH one cannot decide consistency of Π in time $2^{o(k \cdot \log(k))} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$.*

While according to Proposition 1, we cannot expect to significantly improve the runtime for normal logic programs in the worst case, it still is worth studying the underlying reason that makes the worst case so bad. It is well-known that positive cycles are responsible for the hardness (Lifschitz and Razborov 2006; Janhunen 2006) of computing answer sets of normal logic programs. The particular issue with logic programs Π in combination with treewidth and large cycles is that in a TD of G_Π it might be the case that the cycle spreads across the whole decomposition, i.e., TD bags only contain parts of such cycles, which makes it impossible to view these cycles (and dependencies) as a whole. This is also the reason of the hardness given in Proposition 1 and explains why under bounded treewidth evaluating normal logic programs is harder than evaluating propositional formulas. However, if a given normal program only has positive cycles of length at most 3, and each atom appears in at most one positive cycle, the properties of TDs ensure that the atoms of each such positive cycle appear in at least one common bag. Indeed, a cycle of length at most 3 forms a completely connected subgraph and therefore it is guaranteed (Kloks 1994) that the atoms of the cycle are in one common bag of any TD of G_Π.

Example 4. *Recall program Π of Example 1. Observe that in any TD of G_Π it is required that there are nodes t, t' with $\chi(t) \subseteq \{b, c, d\}$ and $\chi(t') \subseteq \{a, b, d\}$ since a cycle of length 3 in the positive dependency graph D_Π (cf. Figure 1) is completely connected in G_Π, cf. Figure 2 (left).*

In the following, we study cycles of length at most ℓ, where we bound the size of these positive cycles in order to obtain results better than the lower bound of Proposition 1 on programs of bounded positive cycle lengths. This provides a significant improvement in the running time on programs, where the size of positive cycles is bounded, and also shows that indeed the case of positive cycle lengths up to 3 can be still efficiently lifted to lengths beyond 3. Consequently, not all positive cycles are bad assuming that the maximum size ℓ of the positive cycles is bounded, thereby obtaining runtimes below the lower bound of Proposition 1 as long as $\ell \ll k$, where k is the treewidth of G_Π.

Bounding Positive Cycles. In the remainder of this work, we assume an HCF logic program Π, whose treewidth is given by $k = tw(G_\Pi)$. We let $\ell_{\mathrm{scc}(a)} \geq 1$ for each atom a be the *number of atoms (size) of the SCC* of a in D_Π. Further, we let $\ell := \max_{a \in \mathrm{at}(\Pi)} \ell_{\mathrm{scc}(a)}$ be the *largest SCC size*. This also bounds the lengths of positive cycles. If each atom a appears in at most one positive cycle, we have that $\ell_{\mathrm{scc}(a)}$ is the cycle length of a and then ℓ is the length of the largest cycle in Π. We refer to the class of HCF logic programs, whose largest SCC size is bounded by a parameter ℓ by *SCC-bounded* ASP. Observe that the largest SCC size ℓ is orthogonal to treewidth.

Example 5. *Consider program Π from Example 1. Then, $\ell_{\mathrm{scc}(e)}{=}\ell_{\mathrm{scc}(f)}{=}\ell_{\mathrm{scc}(g)}{=}1$, $\ell_{\mathrm{scc}(a)}{=}\ell_{\mathrm{scc}(b)}{=}\ell_{\mathrm{scc}(c)}{=}\ell_{\mathrm{scc}(d)}{=}4$, and $\ell{=}4$. Assume a program Π', whose primal graph equals the dependency graph, which is just one large (positive) cycle. This program has treewidth 2 and one can define a TD of $G_{\Pi'}$, whose bags are constructed along the cycle. However, the largest SCC size $\ell = |\mathrm{at}(\Pi')|$. Conversely, there are programs of large treewidth with no positive cycle.*

Bounding sizes of SCCs seems similar to the non-parameterized context, where the consistency of normal logic programs is compiled to a propositional formula (SAT) by a reduction based on level mappings that is applied on an SCC-by-SCC basis (Janhunen 2006). However, this reduction does not preserve the treewidth. On the other hand, while our approach also uses level mappings and proceeds on an SCC-by-SCC basis, the overall evaluation is not SCC-based, since this might completely destroy the treewidth in the worst-case. Instead, the evaluation is guided along a TD.

4 Treewidth-Aware Reductions for SCC-bounded ASP

Next, we present a treewidth-aware reduction from HCF ASP to tight ASP. More precisely, for an HCF program Π with largest SCC size ℓ, where k is the treewidth of G_Π, the resulting tight program has treewidth $\mathcal{O}(k \cdot \log(\ell))$.

Reduction to Tight ASP

The overall construction of the reduction is inspired by the idea of treewidth-aware reductions (Hecher 2020), where in the following, we assume an SCC-bounded program Π and a TD $\mathcal{T} = (T, \chi)$ of G_Π such that the construction of the

resulting tight logic program Π' is heavily guided along $\mathcal{T}$. In contrast to existing work (Hecher 2020), bounding cycles with the largest SCC size additionally allows to have a "global" level mapping (Janhunen 2006), i.e., we do not have different levels for an atom in different bags. Then, while the overall reduction is still guided along the TD $\mathcal{T}$ in order to only slightly increase treewidth, these global level mappings ensure that the tight program is guaranteed to bijectively preserve all answer sets, as stated in Theorem 1.

Before we discuss the construction in detail, we require auxiliary atoms and notation as follows. In order to *guide* the evaluation of the provability of an atom $x \in \text{at}(\Pi)$ in a node t in T along the decomposition $\mathcal{T}$, we use atoms p_t^x and $p_{\leq t}^x$ to indicate that x was proven in node t (with some rule in Π_t) and below t, respectively. Further, we require level mappings similar to related work (Janhunen 2006), but adapted to SCC-bounded programs. Formally, a *level mapping* $\sigma : A \to \{0, \ldots, \ell{-}1\}$ for set of atoms $A \subseteq \text{at}(\Pi)$ is a function mapping each atom $x \in A$ to a level $\sigma(x)$ such that the level does not exceed $\ell_{\text{scc}(x)}$, i.e., $\sigma(x) < \ell_{\text{scc}(x)}$. We use atoms b_x^j, called *level bits*, for $x \in \text{at}(\Pi)$ and $1 \leq j \leq \lceil \log(\ell_{\text{scc}(x)}) \rceil$, which are used as bits in order to represent in a level mapping the level of x in binary. To this end, we denote for x and a number i with $0 \leq i < \ell_{\text{scc}(x)}$ as well as a position number $1 \leq j \leq \lceil \log(\ell_{\text{scc}(x)}) \rceil$, the *$j$-th position of i in binary* by $[i]^j$. Then, we let $[\![x]\!]_i$ be the consistent *set of literals over level bits b_x^j* that is used to represent level number i for x in binary. More precisely, for each position number j, $[\![x]\!]_i$ contains b_x^j if $[i]^j = 1$ and $\neg b_x^j$ otherwise, i.e., if $[i]^j = 0$. Finally, we use auxiliary atoms of the form $x \prec_t^r i$ to indicate that the level for x represented by $[\![x]\!]_i$ is indeed smaller than $i > 0$, which is used in the context of node t and a rule $r \in \Pi_t$ for technical reasons. However, we omit r and t in "$\prec_t^r$" if clear from the context.

Example 6. *Recall program Π, level mapping σ, and largest SCC size $\ell = 4$ from Example 1. For representing σ in binary, we require $\lceil \log(\ell) \rceil = 2$ bits per atom $a \in \text{at}(\Pi)$ and we assume that bits are ordered from least to most significant bit. So $[\sigma(e)]^0 = [\sigma(e)]^1 = 0$, $[\sigma(c)]^0 = 1$ and $[\sigma(c)]^1 = 0$. Then, $[\![e]\!]_{\sigma(e)} = \{\neg b_e^0, \neg b_e^1\}$, $[\![b]\!]_{\sigma(b)} = \{\neg b_b^0, \neg b_b^1\}$, $[\![c]\!]_{\sigma(c)} = \{b_c^0, \neg b_c^1\}$, $[\![d]\!]_{\sigma(d)} = \{\neg b_d^0, b_d^1\}$, and $[\![a]\!]_{\sigma(a)} = \{b_a^0, b_a^1\}$.*

Next, we are ready to discuss the treewidth-aware reduction from SCC-bounded ASP to tight ASP, which takes Π and $\mathcal{T}$ and creates a tight logic program Π'. To this end, let t be any node of T. First, truth values for each atom $x \in \chi(t)$ are subject to a guess by Rules (1) and by Rules (2) it is ensured that all rules of Π_t are satisfied. Notably, by the definition of TDs, Rules (1) and Rules (2) indeed cover all the atoms of Π and all rules of Π, respectively. Then, the next block of rules consisting of Rules (3)–(9) is used for ensuring provability and finally the last block of Rules (10)–(12) is required in order to preserve answer sets, i.e., these rules prevent duplicate answer sets of Π' for an answer set of Π.

For the block of Rules (3)–(9) to ensure provability, we need to guess the level bits for each atom x as given in Rules (3). Rules (4) ensure that we correctly define $x \prec_t^r i$, which is the case if there exists a bit $[i]^j$ that is set to 1, but we have $\neg b_x^j$ and for all larger bits $[i]^{j'}$ that are set to 0 ($j' > j$), we also have $\neg b_x^{j'}$. Then, for Rules (5) we slightly abuse notation $x \prec_t^r i$ for a set X, where $X \prec_t^r i$ denotes a set of atoms of the form $x \prec_t^r i$ for each $x \in X$. Rules (5) make sure that whenever a rule $r \in \Pi_t$ proves x with the level mapping given by the level bits over atoms in $\chi(t)$, we have provability p_t^x for x in t. However, only for the atoms of the positive body B_r^+ which are also in the same SCC $C = \text{scc}(x)$ as x we need to check that the levels are smaller than the level of x, since by definition of SCCs, there cannot be a positive cycle among atoms of different SCCs. As a result, if there is a rule, where no atom of the positive body is in C, satisfying the rule is enough for proving x. If provability p_t^x holds, we also have $p_{\leq t}^x$ by Rules (6) and provability is propagated from node t' to its parent node t by setting $p_{\leq t}^x$ if $p_{\leq t'}^x$, as indicated by Rules (7). Finally, whenever an atom x is not present above a node t, we require provability $p_{\leq t}^x$, ensured by Rules (8) and (9).

Preserving answer sets (bijectively): The last block consisting of Rules (10), (11), and (12) makes sure that atoms that are false or not in the answer set of Π' get level 0 and that we prohibit levels for an atom x that can be decreased by one without losing provability. This ensures that for each answer set of Π we get exactly one answer set of Π' and vice versa.

$$\{x\} \leftarrow \qquad \text{for each } x \in \chi(t); \text{ see}^1 \tag{1}$$

$$\leftarrow B_r^+, \overline{B_r^- \cup H_r} \qquad \text{for each } r \in \Pi_t \tag{2}$$

$$\{b_x^j\} \leftarrow \qquad \text{for each } x \in \chi(t), 1 \leq j \leq \lceil \log(\ell_{\text{scc}(x)}) \rceil; \text{ see}^1 \tag{3}$$

$$x \prec_t^r i \leftarrow \neg b_x^j, \overline{B} \qquad \text{for each } r \in \Pi_t, x \in H_r, C{=}\,\text{scc}(x), 1{\leq}i{<}\ell_C, 1{\leq}j{\leq}\lceil \log(\ell_C) \rceil, [i]^j{=}1, B{=}\{b_x^s \mid j{<}s{\leq}\lceil \log(\ell_C) \rceil, [i]^s{=}0\} \tag{4}$$

$$p_t^x \leftarrow x, [\![x]\!]_i, B_r^+, \overline{B_r^- \cup (H_r \setminus \{x\})}, (B_r^+ \cap \text{scc}(x)) \prec_t^r i \qquad \text{for each } r \in \Pi_t, x \in H_r, 1 \leq i < \ell_{\text{scc}(x)} \tag{5}$$

$$p_{\leq t}^x \leftarrow p_t^x \qquad \text{for each } x \in \chi(t) \tag{6}$$

$$p_{\leq t}^x \leftarrow p_{\leq t'}^x \qquad \text{for each } x \in \chi(t) \cap \chi(t'), t' \in \text{chld}(t) \tag{7}$$

$$\leftarrow x, \neg p_{\leq t'}^x \qquad \text{for each } x \in \chi(t') \setminus \chi(t), t' \in \text{chld}(t) \tag{8}$$

$$\leftarrow x, \neg p_{\leq \text{root}(T)}^x \qquad \text{for each } x \in \chi(\text{root}(T)) \tag{9}$$

$$\leftarrow \neg x, b_x^j \qquad \text{for each } x \in \chi(t), 1 \leq j \leq \lceil \log(\ell_{\text{scc}(x)}) \rceil \tag{10}$$

$$\leftarrow x, [\![x]\!]_i, B_r^+, \overline{B_r^- \cup (H_r \setminus \{x\})}, (B_r^+ \cap C) \prec_t^r i{-}1 \qquad \text{for each } r \in \Pi_t, x \in H_r, C{=}\,\text{scc}(x), 2 \leq i < \ell_C, B_r^+ \cap C \neq \emptyset \tag{11}$$

$$\leftarrow x, [\![x]\!]_i, B_r^+, \overline{B_r^- \cup (H_r \setminus \{x\})} \qquad \text{for each } r \in \Pi_t, x \in H_r, C{=}\,\text{scc}(x), 1 \leq i < \ell_C, B_r^+ \cap C = \emptyset \tag{12}$$

Example 7. *Recall program Π of Example 1 and TD $\mathcal{T} = (T, \chi)$ of G_Π as given in Figure 2. Rules (1) and Rules (2) are constructed for each atom $a \in \text{at}(\Pi)$ and for each rule $r \in \Pi$,*

[1] We view a *choice rule* $\{a\} \leftarrow$ as $a \vee a' \leftarrow$ for a fresh atom a'.

respectively. Similarly, Rules (3) are constructed for each of the $\lceil \log(\ell_{\mathrm{scc}(a)}) \rceil$ *many bits of each atom* $a \in \mathrm{at}(\Pi)$*. Rules (4) serve as auxiliary definition, where for, e.g., atom* c *we construct* $c{\prec}1 \leftarrow \neg b_c^0, \neg b_c^1$*;* $c{\prec}2 \leftarrow \neg b_c^1$*;* $c{\prec}3 \leftarrow \neg b_c^0$*; and* $c{\prec}3 \leftarrow \neg b_c^1$*. Next, we show Rules (5)–(12) for node* t_2*.*

No.	*Rules*
(5)	$p^b_{t_2} \leftarrow b, [\![b]\!]_1, d{\prec}1, d;\ p^b_{t_2} \leftarrow b, [\![b]\!]_2, d{\prec}2, d;$
	$p^b_{t_2} \leftarrow b, [\![b]\!]_3, d{\prec}3, d;$
	$p^c_{t_2} \leftarrow c, [\![c]\!]_1, d{\prec}1, d;\ p^c_{t_2} \leftarrow c, [\![c]\!]_2, d{\prec}2, d;$
	$p^c_{t_2} \leftarrow c, [\![c]\!]_3, d{\prec}3, d;$
	$p^d_{t_2} \leftarrow d, [\![d]\!]_1, b{\prec}1, c{\prec}1, b, c;\ p^d_{t_2} \leftarrow d, [\![d]\!]_2, b{\prec}2, c{\prec}2,$
	$b, c;\ p^d_{t_2} \leftarrow d, [\![d]\!]_3, b{\prec}3, c{\prec}3, b, c$
(6)	$p^b_{\leq t_2} \leftarrow p^b_{t_2};\ p^c_{\leq t_2} \leftarrow p^c_{t_2};\ p^d_{\leq t_2} \leftarrow p^d_{t_2}$
(10)	$\leftarrow \neg b, b^0_b;\ \leftarrow \neg b, b^1_b;\ \leftarrow \neg c, b^0_c;\ \leftarrow \neg c, b^1_c;$
	$\leftarrow \neg d, b^0_d;\ \leftarrow \neg d, b^1_d$
(11)	$\leftarrow b, [\![b]\!]_2, d{\prec}1, d;\ \leftarrow b, [\![b]\!]_3, d{\prec}2, d;$
	$\leftarrow c, [\![c]\!]_2, d{\prec}1, d;\ \leftarrow c, [\![c]\!]_3, d{\prec}2, d;$
	$\leftarrow d, [\![d]\!]_2, b{\prec}1, c{\prec}1, b, c;\ \leftarrow d, [\![d]\!]_3, b{\prec}2, c{\prec}2, b, c$

For root node t_5 *of* T*, we obtain the following Rules (5)–(12).*

(5)	$p^b_{t_5} \leftarrow b, e, \neg f$ *(simplified since* $B^+_{r_4} \cap \mathrm{scc}(b) = \emptyset$*)*
(6)	$p^b_{\leq t_5} \leftarrow p^b_{t_5};\ p^e_{\leq t_5} \leftarrow p^e_{t_5};\ p^f_{\leq t_5} \leftarrow p^f_{t_5}$
(7)	$p^b_{\leq t_5} \leftarrow p^b_{\leq t_3};\ p^e_{\leq t_5} \leftarrow p^e_{\leq t_3};\ p^e_{\leq t_5} \leftarrow p^e_{\leq t_4};\ p^f_{\leq t_5} \leftarrow p^f_{\leq t_4}$
(8)	$\leftarrow d, \neg p^d_{\leq t_3};\ \leftarrow g, \neg p^g_{\leq t_4}$
(9)	$\leftarrow b, \neg p^b_{\leq t_5};\ \leftarrow e, \neg p^e_{\leq t_5};\ \leftarrow f, \neg p^f_{\leq t_5}$
(10)	$\leftarrow \neg b, b^0_b;\ \leftarrow \neg b, b^1_b;\ \leftarrow \neg e, b^0_e;\ \leftarrow \neg e, b^1_e;$
	$\leftarrow \neg f, b^0_f;\ \leftarrow \neg f, b^1_f$
(12)	$\leftarrow b, [\![b]\!]_1, e, \neg f;\ \leftarrow b, [\![b]\!]_2, e, \neg f;\ \leftarrow b, [\![b]\!]_3, e, \neg f$

Correctness and Treewidth-Awareness. We discuss correctness and treewidth-awareness, leading then to Theorem 1.

Lemma 1 (Correctness)**.** *Let* Π *be an HCF program, where* k *is the treewidth of* G_Π *and every SCC* C *satisfies* $|C| \leq \ell$*. Then, tight program* Π' *obtained by Rules (1)–(12) on* Π *and a TD* $\mathcal{T} = (T, \chi)$ *of* G_Π *is correct. Formally, for any answer set* I *of* Π *there is exactly one answer set* I' *of* Π' *(vice versa).*

Proof. "$\Longrightarrow$": Let I be an answer set of Π. Then, there exists a unique (Janhunen 2006), minimal[2] level mapping σ proving each $x \in I$ with $0 \leq \sigma(x) < \ell_{\mathrm{scc}(x)}$. Let $P := \{p^x_t, p^x_{\leq t} \mid r \in \Pi_t \text{ proves } x \text{ with } \sigma, x \in I, t \text{ in } T\}$. From this we construct an interpretation $I' := I \cup \{b^j_x \mid [\sigma(x)]^j = 1, 1 \leq j \leq \lceil \log(\ell_{\mathrm{scc}(x)}) \rceil, x \in I\} \cup P \cup \{p^x_{\leq t} \mid x \in I, t' \in T, t' \text{ is below } t \text{ in } T, p^x_{\leq t'} \in P\} \cup \{x \prec^r_t i \mid t \in T, r \in \Pi_t, x \in H_r, i < \ell_{\mathrm{scc}(x)}, \sigma(x) < i\}$, which sets atoms as I, encodes σ in binary and sets provability accordingly. It is easy to see that I' is an answer set of Π'. "$\Longleftarrow$": Let I' be an answer set of Π'. From this we construct $I := I' \cap \mathrm{at}(\Pi)$ as well as level mapping $\sigma := \{x \mapsto f_{I'}(x) \mid x \in \mathrm{at}(\Pi)\}$, where we define function $f_{I'}(x) : \mathrm{at}(\Pi) \to \{0, \ldots, \ell{-}1\}$ for atom $x \in \mathrm{at}(\Pi)$ to return $0 \leq i < \ell_{\mathrm{scc}(x)}$ if $\{b^j_x \mid 1 \leq j \leq \lceil \log(\ell_{\mathrm{scc}(x)}) \rceil, [i]^j = 1\} = \{b^j_x \in I' \mid 1 \leq j \leq \lceil \log(\ell_{\mathrm{scc}(x)}) \rceil\}$, i.e., I' binary-encodes i for x. Assume towards a contradiction that $I \not\models \Pi$. But then I' does not satisfy at least one instance of Rules (1) and (2), contradicting that I' is an answer set of Π'. Again, towards a contradiction assume that I is not an answer set of Π, i.e., at least one $x \in \mathrm{at}(\Pi)$ cannot be proven with σ. We still have $p^x_{\leq \mathrm{root}(T)} \in I'$, by Rules (8) and (9). However, then we either have that $p^x_{\leq t} \in I'$ or $p^x_{\mathrm{root}(T)} \in I'$ by Rules (6) and (7) for at least one child node t of $\mathrm{root}(T)$. Finally, by connectedness property (iii) of TDs, there has to be a node t' that is either $\mathrm{root}(T)$ or a descendant of $\mathrm{root}(T)$ where we have $p^x_{t'} \in I'$. Consequently, by Rules (5) as well as Rules (3) and (4) we have a rule $r \in \Pi$ that proves x with σ, contradicting the assumption. Similarly, Rules (10), (11), and (12) ensure minimality of σ. □

Lemma 2 (Treewidth-Awareness)**.** *Let* Π *be an HCF program s.t. every SCC* C *satisfies* $|C| \leq \ell$*. Then, the treewidth of tight program* Π' *obtained by the reduction above on* Π *and a TD* $\mathcal{T} = (T, \chi)$ *of* G_Π *of width* k*, is in* $\mathcal{O}(k \cdot \log(\ell))$*.*

Proof (Sketch). We take $\mathcal{T}{=}(T, \chi)$ and construct a TD $\mathcal{T}' {:=} (T', \chi')$ of $G_{\Pi'}$, where T' is obtained by replacing each node t of T by a sequence of nodes. The sequence for t consists for each $r \in \Pi_t$ of ℓ many nodes $t^r_1, \ldots, t^r_\ell$, where χ' is defined as follows. For each node t^r_i s.t. the parent of t in T is t^*, we let $\chi'(t^r_i) := \chi(t) \cup \{b^j_x \mid x \in \chi(t), 1 \leq j \leq \lceil \log(\ell_{\mathrm{scc}(x)}) \rceil\} \cup \{p^x_t, p^x_{\leq t}, p_{\leq t^*} \mid x \in \chi(t)\} \cup \{y \prec^r_t i' \mid y \in B^+_r \cap \mathrm{scc}(x), x \in H_r, i' \in \{i{-}1, i\}, 1 \leq i' < \ell_{\mathrm{scc}(x)}\}$. Indeed, all atoms of every instance of Rules (1)–(12) appear in at least one common bag of χ'. Further, $\mathcal{T}'$ is connected, i.e., $\mathcal{T}'$ is a TD of $G_{\Pi'}$ and $|\chi'(t)|$ in $\mathcal{O}(k \cdot \log(\ell))$. □

Theorem 1 (Removing Cyclicity of SCC-bounded ASP)**.** *Let* Π *be an HCF program, where the treewidth of* G_Π *is at most* k *and every SCC* C *satisfies* $|C| \leq \ell$*. Then, there is a tight program* Π' *with treewidth in* $\mathcal{O}(k \cdot \log(\ell))$ *such that for each answer set* M *of* Π *there is exactly one answer set of* Π' *that is equal to* M *over atoms* $\mathrm{at}(\Pi)$*, and vice versa.*

Proof. First, we compute a TD $\mathcal{T} = (T, \chi)$ of G_Π of width below $5 \cdot k$ ("5-*approximation*"), where $k = tw(G_\Pi)$, in time $2^{\mathcal{O}(k)} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$. Observe that the reduction consisting of Rules (1)–(12) on Π and $\mathcal{T}$ runs in time $\mathcal{O}(k \cdot \ell \cdot \log(\ell)^2 \cdot (|\mathrm{at}(\Pi)| + |\Pi|))$. The claim follows by correctness (Lemma 1) and by treewidth-awareness of Lemma 2. □

Having established Theorem 1, the reduction above allows to define an algorithm for computing answer sets of Π, running in time $2^{\mathcal{O}(k \cdot \log(\lambda))} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$, where $\lambda = \min(\{k, \ell\})$. This can be achieved by compiling the resulting tight program of the reduction above to a propositional formula (SAT), which runs in single-exponential time in k, cf. Section 4. Then, we use an algorithm for SAT running in time single-exponential in the treewidth (Samer and Szeider 2010). In contrast to existing work (Fichte and Hecher 2019) the overall approach bijectively preserves the answer sets of Π, which enables counting or enumeration of all answer sets with linear delay. Notably, if largest SCC size $\ell \ll k$ we improve known runtimes (cf. Proposition 1).

Reduction to Almost Tight ASP– Balancing Treewidth and Tightness Width

Now, we focus on the consistency problem for SCC-bounded ASP, since relaxing the (bijective) preservation of answer sets allows us to further improve our reduction. Intuitively,

[2] σ is minimal if no level can be decreased s.t. I is proven with σ.

with "local" level mappings which provide level numbers only locally for each TD bag, but ensure compatibility among different bags, one might obtain duplicate answer sets. While these local mappings for TD bags were already used (Hecher 2020), applying local mappings for SCC-bounded ASP enables significant improvements which motivate a new measure for (relaxed variants) of tightness for ASP and treewidth.

Towards Tightness Width and Almost Tightness. To this end, let Π be an SCC-bounded program, $\mathcal{T} = (T, \chi)$ be a TD of G_Π of width k, t be a node of T, and ℓ be the largest SCC size of Π. Then, we let $\ell_t^{\mathrm{scc}(x)}$ for each atom x of Π be the *size of the SCC* of x restricted to $\chi(t)$ in D_Π, i.e., $\ell_t^{\mathrm{scc}(x)} := |\chi(t) \cap \mathrm{scc}(x)|$. Further, we define the *local SCC size* ℓ_t for a node t as follows $\ell_t := \max_{x \in \chi(t)} \ell_t^{\mathrm{scc}(x)}$ and the *tightness width* ι by $\iota := \max_{t \text{ of } T} \ell_t$. Intuitively, the local SCC sizes are bounded by the bag sizes $(k+1)$ as well as the largest SCC size ℓ, but can be even significantly smaller, especially if SCCs are spread across $\mathcal{T}$ and if TD bags contain parts of many SCCs. This observation motivates a new (relaxed) measure for almost tightness of ASP and treewidth, where even non-tight ASP programs with large cycles are considered almost tight on $\mathcal{T}$, as long as ι is small (constant). We say program Π is *ι-tight on $\mathcal{T}$*, i.e., "almost" tight, if $\iota \geq 2$. Notably, if $\iota = 1$, Π is tight and vice versa.

Example 8. *Recall program Π of Example 1 and TD $\mathcal{T} = (T, \chi)$ of G_Π of Figure 2. Observe that Π is 3-tight on $\mathcal{T}$. Consider program Π' from Example 5. While ℓ is large ($\ell = |\mathrm{at}(\Pi')|$), we have that $\iota = 3$ for any TD of $G_{\Pi'}$ of width 2.*

Increasing Almost Tightness (Decreasing ι). Below, we provide a reduction that reduces the tightness width of almost tight programs. To this end, we assume a program Π being ι-tight on a TD $\mathcal{T} = (T, \chi)$ of G_Π, whose width k coincides with the treewidth of G_Π, and a node t of T. The reduction indirectly relies on *local level mappings* $\sigma_t : A \to \{0, \ldots, \ell_t - 1\}$ for set of atoms $A \subseteq \mathrm{at}(\Pi)$, which is a function mapping each atom $x \in A$ to a level $\sigma(x)$ such that the level does not exceed $\ell_t^{\mathrm{scc}(x)}$, i.e., $\sigma(x) < \ell_t^{\mathrm{scc}(x)}$. However, we require atoms $b_{t,x}^j$, called *local level bits*, for $x \in \chi(t)$ and $1 \leq j \leq \lceil \log(\ell_t^{\mathrm{scc}(x)}) \rceil$, for representing level $\sigma_t(x)$ of x for node t in binary. Analogously to above, we denote the *j-th position of i for t in binary* with $0 \leq i < \ell_t^{\mathrm{scc}(x)}$ by $[i]_t^j$, as well as the consistent *literals over local level bits* $b_{t,x}^j$ for representing level i of x for t by $[\![x]\!]_{t,i}$. As before, we use auxiliary atoms of the form $x \prec_t^r i$ to indicate $\sigma_t(x) < i$ for $x \in H_r$ with $r \in \Pi_t$, as well as provability atoms p_t^x, $p_{\leq t}^x$.

We are ready to discuss the treewidth-aware reduction from program Π being ι-tight on $\mathcal{T}$ to a program Π' that is ι'-tight on a TD $\mathcal{T}'$ with $\iota' \leq \iota$, where $\mathcal{T}'$ is constructed as in Lemma 2. To this end, let $\mathcal{C}$ be the union of those non-trivial SCCs of D_Π that shall be eliminated. If $\mathcal{C} = \emptyset$, we obtain the program Π as result, and if $\mathcal{C}$ are all vertices of any non-trivial SCC of D_Π, we obtain a tight program, whose treewidth is bounded by $\mathcal{O}(k \cdot \log(\iota))$. Consequently, if $\mathcal{C}$ are the vertices of those non-trivial SCCs that are responsible for large local SCC sizes, one can increase the level of tightness (decrease ι) at the cost of a slight increase of treewidth. The reduction consists of Rules (13)–(23). First, truth values for each atom $x \in \chi(t) \cap \mathcal{C}$ are guessed by Rules (13) and by Rules (14) it is ensured that Π_t is satisfied and that those atoms in the head H_r of a rule $r \in \Pi_t$ but not in $\mathcal{C}$ still appear in the head. The next block of Rules (15)–(18) is used for guessing local level bits for atoms $x \in \chi(t) \cap \mathcal{C}$, cf. Rules (15) and defining auxiliary atom $x \prec_t^r i$ by Rules (16). Further, by Rules (17) and (18) it is ensured that the different local level mappings for different nodes are compatible. More concretely, two neighboring nodes of T do not order two atoms of the same SCC differently. Finally, Rules (19)–(23) ensure provability similar to Rules (5)–(9), but only for atoms in $\mathcal{C}$ as well as rules, whose heads contain atoms of $\mathcal{C}$.

$$\{x\} \leftarrow \quad \text{for each } x \in \chi(t) \cap \mathcal{C}; \text{ see}^1 \tag{13}$$

$$H_r \backslash \mathcal{C} \leftarrow B_r^+, \quad \overline{B_r^-} \cup (H_r \cap \mathcal{C}) \quad \text{for each } r \in \Pi_t \tag{14}$$

$$\{b_{t,x}^j\} \leftarrow \quad \text{for each } r \in \Pi_t, x \in H_r \cap \mathcal{C}, 1 \leq j \leq \lceil \log(\ell_t^{\mathrm{scc}(x)}) \rceil; \text{ see}^1 \tag{15}$$

$$x \prec_t^r i \leftarrow \neg b_{t,x}^j, \overline{B} \quad \text{for each } r \in \Pi_t, x \in H_r \cap \mathcal{C}, C = \mathrm{scc}(x), 1 \leq i < \ell_t^C, 1 \leq j \leq \lceil \log(\ell_t^C) \rceil, [i]_t^j = 1, B = \{b_{t,x}^s \mid j < s \leq \lceil \log(\ell_t^C) \rceil, [i]_t^s = 0\} \tag{16}$$

$$\leftarrow b_{t,x}^1 = b_{t,y}^1, \ldots, b_{t,x}^l = b_{t,y}^l \quad \text{for each } x, y \in \chi(t) \cap \mathcal{C}, C = \mathrm{scc}(x), y \in C, x \neq y, l = \lceil \log(\ell_t^C) \rceil \tag{17}$$

$$\leftarrow (x \dot{\prec}_t^j y), (y \dot{\prec}_{t'}^{j'} x) \quad \text{for each } x, y \in \chi(t) \cap \chi(t') \cap \mathcal{C}, t' \in \mathrm{chld}(t), C = \mathrm{scc}(x), 1 \leq j \leq \lceil \log(\ell_t^C) \rceil, y \in C, 1 \leq j' \leq \lceil \log(\ell_{t'}^C) \rceil; \text{ see}^3 \tag{18}$$

$$p_t^x \leftarrow x, B_r^+, [\![x]\!]_{t,i} \quad \overline{B_r^-} \cup (H_r \backslash \{x\}), (B_r^+ \cap C) \prec_t^r i \quad \text{for each } r \in \Pi_t, x \in H_r \cap \mathcal{C}, C = \mathrm{scc}(x), 1 \leq i < \ell_t^C \tag{19}$$

$$p_{\leq t}^x \leftarrow p_t^x \quad \text{for each } x \in \chi(t) \cap \mathcal{C} \tag{20}$$

$$p_{\leq t}^x \leftarrow p_{\leq t'}^x \quad \text{for each } x \in \chi(t) \cap \chi(t') \cap \mathcal{C}, t' \in \mathrm{chld}(t) \tag{21}$$

$$\leftarrow x, \neg p_{\leq t'}^x \quad \text{for each } x \in (\chi(t') \cap \mathcal{C}) \setminus \chi(t), t' \in \mathrm{chld}(t) \tag{22}$$

$$\leftarrow x, \neg p_{\leq \mathrm{root}(T)}^x \quad \text{for each } x \in \chi(\mathrm{root}(T)) \cap \mathcal{C} \tag{23}$$

Treewidth-Awareness and Consequences. The reduction above allows us to show Lemma 3 and Theorem 2, whose proofs are similar to Lemma 2 and Theorem 1, respectively.

Lemma 3 (Treewidth-Awareness)**.** *Let Π be a program that is ι-tight on a TD $\mathcal{T}$ of width k. Then, the treewidth of tight program Π' obtained by Rules (13)–(23) on Π, $\mathcal{T}$, and set $\mathcal{C}$ of vertices of all non-trivial SCCs of D_Π, is in $\mathcal{O}(k \cdot \log(\iota))$.*

Theorem 2 (Removing Cyclicity of ι-tight ASP)**.** *Let Π be a program being ι-tight on a TD $\mathcal{T}$ of width k. Then, there is a tight program Π' with treewidth in $\mathcal{O}(k \cdot \log(\iota))$ such that the answer sets of Π and Π' projected to $\mathrm{at}(\Pi)$ coincide.*

Reduction to SAT

Next, we present a treewidth-aware reduction from tight ASP to SAT, which together with the previous reduction allows

[3] Let $(x \dot{\prec}_t^j y)$ be $\neg b_{t,x}^j, b_{t,y}^j, b_{t,x}^{j+1} \leq b_{t,y}^{j+1}, \ldots, b_{t,x}^{\ell_{\mathrm{scc}(x)}} \leq b_{t,y}^{\ell_{\mathrm{scc}(x)}}$.

us to reduce from SCC-bounded ASP to SAT. While the step from tight ASP to SAT might seem straightforward for the program obtained by one of the reductions above, in general it is not guaranteed that existing reductions, e.g., (Fages 1994; Lin and Zhao 2003; Janhunen 2006), do not cause a significant blowup in the treewidth.

Let Π be any given tight logic program and $\mathcal{T} = (T, \chi)$ be a tree decomposition of G_Π. Similar to the reductions from SCC-bounded ASP to tight ASP, we use as variables besides the original atoms of Π also auxiliary variables. In order to preserve treewidth, we still need to guide the evaluation of the provability of an atom $x \in \text{at}(\Pi)$ in a node t in T along the TD $\mathcal{T}$, whereby we use atoms p^x_t and $p^x_{\leq t}$ to indicate that x was proven in node t and below t, respectively. However, we do not need any level mappings, since there is no positive cycle in Π, but we still guide provability, cf. Clark's completion (Clark 1977), along TD $\mathcal{T}$. Consequently, we construct the following propositional formula, where for each node t of T we add Formulas (24)–(28). Intuitively, Formulas (24) ensure that all rules are satisfied, cf. Rules (2). Formulas (25) and (26) take care that ultimately an atom that is set to true must be proven, similar to Rules (8) and (9). Finally, Formulas (27) and (28) provide the definition for an atom to be proven in a node and below a node, respectively, which is similar to Rules (5)–(7), but without the level mappings.

Preserving answer sets: We obtain exactly one model of the resulting formula for each answer set of Π. This can be weakened by turning equivalences ($\leftrightarrow$) into implications ($\rightarrow$).

$$\bigvee_{a \in B^+_r} \neg a \vee \bigvee_{a \in B^-_r \cup H_r} a \qquad \text{for each } r \in \Pi_t \tag{24}$$

$$x \rightarrow p^x_{\leq t'} \qquad \text{for each } t' \in \text{chld}(t),\ x \in \chi(t') \setminus \chi(t) \tag{25}$$

$$x \rightarrow p^x_{\leq \text{root}(T)} \qquad \text{for each } x \in \chi(\text{root}(T)) \tag{26}$$

$$p^x_t \leftrightarrow \bigvee_{r \in \Pi_t, x \in H_r} \Big(\bigwedge_{a \in B^+_r} a \wedge x \wedge \bigwedge_{b \in B^-_r \cup (H_r \setminus \{x\})} \neg b \Big) \qquad \text{for each } x \in \chi(t) \tag{27}$$

$$p^x_{\leq t} \leftrightarrow p^x_t \vee \Big(\bigvee_{t' \in \text{chld}(t), x \in \chi(t')} p^x_{\leq t'} \Big) \qquad \text{for each } x \in \chi(t) \tag{28}$$

Correctness and Treewidth-Awareness. Conceptually the proofs of Lemmas 4 and 5 proceed similar to the proofs of Lemmas 1 and 2, but without level mappings, respectively.

Lemma 4 (Correctness)**.** *Let Π be a tight logic program, where the treewidth of G_Π is at most k. Then, the propositional formula F obtained by the reduction above on Π and a TD $\mathcal{T}$ of primal graph G_Π, consisting of Formulas (24)–(28), is correct. Formally, for any answer set I of Π there is exactly one satisfying assignment of F and vice versa.*

Lemma 5 (Treewidth-Awareness)**.** *Let Π be a tight logic program. Then, the treewidth of propositional formula F obtained by the reduction above, consisting of Formulas (24)–(28), by using Π and a TD $\mathcal{T}$ of G_Π of width k is in $\mathcal{O}(k)$.*

However, we cannot do much better, as shown next.

Proposition 2 (ETH-Tightness)**.** *Let Π be a tight logic program, where the treewidth of G_Π is at most k. Then, under ETH, one cannot reduce Π to propositional formula F in time $2^{o(k)} \cdot \text{poly}(|\text{at}(\Pi)|)$ such that $tw(G_F)$ is in $o(k)$.*

Proof. First, we reduce SAT to tight ASP, i.e., capture all models of a given formula F in a tight program Π. Thereby Π consists of a choice rule for each variable of F and a constraint for each clause. Towards a contradiction assume the contrary of this proposition. Then, we reduce Π back to a propositional formula F', running in time $2^{o(k)} \cdot \text{poly}(|\text{at}(\Pi)|)$ with $tw(G_{F'})$ being in $o(k)$. Hence, we use an algorithm for SAT (Samer and Szeider 2010) on F' to solve F with n variables in time $2^{o(k)} \cdot \text{poly}(|n|)$, which contradicts ETH. □

Knowing that under ETH tight ASP has about the same complexity for treewidth as SAT, cf. Proposition 2, we can derive the following corollary, which completes Proposition 1.

Corollary 1. *Let Π be any normal logic program, where the treewidth of G_Π is at most k. Then, under ETH, one cannot reduce Π to a tight logic program Π' in time $2^{o(k \cdot \log(k))} \cdot \text{poly}(|\text{at}(\Pi)|)$ such that $tw(G_{\Pi'})$ is in $o(k \cdot \log(k))$.*

The consistency of a program Π that is ι-tight on $\mathcal{T}$ can be decided in a runtime that is similar to the runtime of SAT for small ι, which improves Proposition 1 as follows.

Theorem 3 (Runtime of ι-tight ASP)**.** *Assume a program Π that is ι-tight on a TD $\mathcal{T}$ of width k, whose number of nodes is linear in $|\text{at}(\Pi)|$. Then, there is an algorithm for deciding the consistency of Π, running in time $2^{\mathcal{O}(k \cdot \log(\iota))} \cdot \text{poly}(|\text{at}(\Pi)|)$.*

Proof. First, we apply the reduction of Section 4 on Π and $\mathcal{T}$ on the set $\mathcal{C}$ of all non-trivial SCCs of D_Π. This results in a tight program, which is reduced by the reduction of Section 4 to obtain a propositional formula F. Both reductions run in time $\mathcal{O}(k^2 \cdot \iota^2 \cdot \log(\iota) \cdot (|\text{at}(\Pi)| + |\Pi|))$. Finally, formula F, whose treewidth is in $\mathcal{O}(k \cdot \log(\iota))$ by Lemmas 3 and 5, are solved by an algorithm (Samer and Szeider 2010) for SAT in time $2^{\mathcal{O}(k \cdot \log(\iota))} \cdot (|\text{at}(\Pi)| + |\Pi|)$. □

Theorem 3 assumes a given TD, efficiently computable by means of heuristics (Abseher, Musliu, and Woltran 2017). Alternatively, one can compute (Bodlaender et al. 2016) a TD of G_Π of width below $5 \cdot tw(G_\Pi)$ (5-approximation) that has a number of nodes linear in $|\text{at}(\Pi)|$, in time $2^{\mathcal{O}(k)} \cdot |\text{at}(\Pi)|$.

5 Conclusion and Future Work

This paper deals with improving algorithms for deciding consistency of head-cycle-free (HCF) ASP of bounded treewidth. Existing works imply that under the exponential time hypothesis (ETH), one cannot solve an HCF program with n atoms and treewidth k in time $2^{o(k \cdot \log(k))} \cdot \text{poly}(n)$.

In this work, we also consider the size ℓ of the largest SCC of the positive dependency graph, yielding a more precise characterization of the runtime: $2^{\mathcal{O}(k \cdot \log(\iota))} \cdot \text{poly}(n)$, where ι is a novel measure bounded by both k and ℓ for measuring the tightness of a program ($\iota = 1$ for tight programs only). Further, we provide a treewidth-aware reduction from HCF ASP to tight ASP, where the treewidth increases from k to $\mathcal{O}(k \cdot \log(\iota))$. Finally, under ETH, tight ASP has similar complexity as SAT: There is no reduction from HCF ASP to tight ASP, increasing treewidth from k to only $o(k \cdot \log(k))$.

Currently, we are performing practical analysis of our provided reductions. For future work we suggest to investigate lower bounds for both parameters k, ι, or under extensions of ETH like strong ETH (Impagliazzo and Paturi 2001).

References

Abseher, M.; Musliu, N.; and Woltran, S. 2017. htd - A Free, Open-Source Framework for (Customized) Tree Decompositions and Beyond. In *CPAIOR'17*, volume 10335 of *LNCS*, 376–386. Springer.

Alviano, M.; Calimeri, F.; Dodaro, C.; Fuscà, D.; Leone, N.; Perri, S.; Ricca, F.; Veltri, P.; and Zangari, J. 2017. The ASP System DLV2. In *LPNMR'17*, volume 10377 of *LNAI*, 215–221. Springer.

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. ISSN 1012-2443. doi:10.1007/BF01530761.

Bichler, M.; Morak, M.; and Woltran, S. 2018. Single-Shot Epistemic Logic Program Solving. In *IJCAI'18*, 1714–1720. ijcai.org.

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.

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 J. Comput.* 45(2): 317–378.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12): 92–103. ISSN 0001-0782. doi:10.1145/2043174.2043195.

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.

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. doi:10.1007/BF01536399.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Methods Log. Comput. Sci.* 1(1): 51–60.

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.; Morak, M.; and Woltran, S. 2017. Answer Set Solving with Bounded Treewidth Revisited. In *LPNMR'17*, volume 10377 of *LNCS*, 132–145. Springer. ISBN 978-3-319-61660-5. doi:10.1007/978-3-319-61660-5_13.

Fichte, J. K.; Kronegger, M.; and Woltran, S. 2019. A multiparametric view on answer set programming. *Ann. Math. Artif. Intell.* 86(1-3): 121–147.

Fichte, J. K.; and Szeider, S. 2015. Backdoors to Tractable Answer-Set Programming. *Artificial Intelligence* 220(0): 64–103. ISSN 0004-3702. doi:10.1016/j.artint.2014.12.001.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. *Answer Set Solving in Practice*. Morgan & Claypool. doi:10.2200/S00457ED1V01Y201211AIM019.

Gelfond, M.; and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.* 9(3/4): 365–386. doi:10.1007/BF03037169.

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. doi:10.1093/bioinformatics/btt393. Erratum see Bioinformatics 30, 13, 1942.

Hecher, M. 2020. Treewidth-Aware Reductions of normal ASP to SAT – Is Normal ASP harder than SAT After All? In *KR'20*, 485–495.

Impagliazzo, R.; and Paturi, R. 2001. On the Complexity of k-SAT. *J. Comput. Syst. Sci.* 62(2): 367–375.

Impagliazzo, R.; Paturi, R.; and Zane, F. 2001. Which Problems Have Strongly Exponential Complexity? *J. of Computer and System Sciences* 63(4): 512–530. ISSN 0022-0000. doi:10.1006/jcss.2001.1774.

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. *Journal of Applied Non-Classical Logics* 16(1-2): 35–86.

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, X. 2004. On Odd and Even Cycles in Normal Logic Programs. In *AAAI*, 80–85. AAAI Press / MIT Press.

Lonc, Z.; and Truszczynski, M. 2003. Fixed-parameter complexity of semantics for logic programs. *ACM Trans. Comput. Log.* 4(1): 91–119.

Marek, W.; and Truszczyński, M. 1991. Autoepistemic logic. *J. of the ACM* 38(3): 588–619. ISSN 0004-5411. doi:10.1145/116825.116836.

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.

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. doi:10.1016/j.jda.2009.06.002.