Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Main Track

Treewidth-aware Reductions of Normal ASP to SAT– Is Normal ASP Harder than SAT after All?

Markus Hecher

TU Wien, Vienna, Austria
University of Potsdam, Potsdam, Germany
hecher@dbai.tuwien.ac.at

Abstract

Answer Set Programming (ASP) is a paradigm and problem modeling/solving toolkit for KR that is often invoked. There are plenty of results dedicated to studying the hardness of (fragments of) ASP. So far, these studies resulted in characterizations in terms of computational complexity as well as in fine-grained insights presented in form of dichotomy-style results, lower bounds when translating to other formalisms like propositional satisfiability (SAT), and even detailed parameterized complexity landscapes. A quite generic and prominent parameter in parameterized complexity originating from graph theory is the so-called *treewidth*, which in a sense captures structural density of a program. Recently, there was an increase in the number of treewidth-based solvers related to SAT. While there exist several translations from (normal) ASP to SAT, yet there is no reduction preserving treewidth or at least being aware of the treewidth increase. This paper deals with a novel reduction from normal ASP to SAT that is aware of the treewidth, and guarantees that a slight increase of treewidth is indeed sufficient. Then, we also present a new result establishing that when considering treewidth, already the fragment of normal ASP is slightly harder than SAT (under reasonable assumptions in computational complexity). This also confirms that our reduction probably cannot be significantly improved and that the slight increase of treewidth is unavoidable.

1 Introduction

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011) is an active research area of knowledge representation and reasoning. ASP provides a declarative modeling language and problem solving framework (Gebser et al. 2012) for hard computational problems, which has been widely applied (Balduccini, Gelfond, and Nogueira 2006; Niemelä, Simons, and Soininen 1999; Nogueira et al. 2001; Guziolowski et al. 2013; Schaub and Woltran 2018). In ASP questions are encoded into rules and constraints that form a program (over atoms), whose solutions are called answer sets.

In terms of computational complexity, the *consistency problem* of deciding the existence of an answer set is well-studied, i.e., the problem is Σ_2^P-complete (Eiter and Gottlob 1995). Some fragments of ASP have lower complexity though. A prominent example is the class of *head-cycle-free (HCF)* programs (Ben-Eliyahu and Dechter 1994), which is a certain generalization of the class of *normal* programs and requires the absence of cycles in a certain graph representation of the program. Deciding whether such a program has an answer set is NP-complete.

There is also a wide range of more fine-grained studies (Truszczynski 2011) for ASP, also in parameterized complexity (Cygan et al. 2015; Niedermeier 2006; Downey and Fellows 2013; Flum and Grohe 2006), where certain (combinations of) parameters (Fichte, Kronegger, and Woltran 2019; Lackner and Pfandler 2012) are taken into account. In parameterized complexity, the "hardness" of a problem is classified according to the impact of a *parameter* for solving the problem. There, one often distinguishes the runtime dependency of the parameter, e.g., levels of exponentiality (Lokshtanov, Marx, and Saurabh 2011; Marx and Mitsou 2016) in the parameter, required for problem solving. Concretely, under the reasonable *Exponential Time Hypothesis (ETH)* (Impagliazzo, Paturi, and Zane 2001), *propositional satisfiability (*SAT*)* is single exponential in the structural parameter treewidth, wheras evaluating *Quantified Boolean formulas (QBFs)* of quantifier depth two is (Lampis and Mitsou 2017) double exponential[1] in the treewidth k.

For ASP there is growing research on treewidth (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017; Fichte and Hecher 2019), which even involves grounding (Bichler, Morak, and Woltran 2018; Bliem et al. 2020). Algorithms of these works exploit structural restrictions (in form of treewidth) of a given program, and often run in polynomial time in the program size, while being exponential only in the treewidth. Intuitively, treewidth gives rise to a *tree decomposition*, which allows solving numerous NP-hard problems in parts, cf., divide-and-conquer, and indicates the maximum number of variables one has to investigate in such parts during evaluation. There were also dedicated competitions (Dell et al. 2017) and notable progresses in SAT (Fichte, Hecher, and Zisser 2019; Charwat and Woltran 2019) and other areas (Bannach and Berndt 2019).

Naturally, there are numerous reductions of ASP (Clark 1977; Ben-Eliyahu and Dechter 1994; Lin and Zhao 2003; Janhunen 2006; Alviano and Dodaro 2016) and extensions thereof (Bomanson and Janhunen 2013; Bomanson 2017) to SAT. These reductions have been investigated in the context of resulting formula size and number of auxiliary variables. However, structural dependency in form of, e.g., treewidth,

[1] Double exponentiality refers to runtimes of the form $2^{2^{\mathcal{O}(k)}} \cdot n$.

has not been considered yet. These existing reductions cause only sub-quadratic blow-up in the number of variables (auxiliary variables), which is unavoidable (Lifschitz and Razborov 2006) if the answer sets should be preserved (bijectively). However, if one considers the structural dependency in form of treewidth, existing reductions could cause quadratic or even unbounded overhead in the treewidth. On the contrary, we present a novel reduction for HCF programs that increases the treewidth k at most *sub-quadratically* ($k \cdot \log(k)$). This is indeed interesting as there is a close connection (Atserias, Fichte, and Thurley 2011) between resolution-width and treewidth, resulting in efficient SAT solver runs on instances of small treewidth. As a result, our reduction could improve solving approaches by means of SAT solvers, e.g., (Lin and Zhao 2004). Then, we establish lower bounds under ETH, for exploiting treewidth for concistency of normal programs. This renders normal ASP "harder" than SAT. At the same time we prove that one can not significantly improve the reduction, i.e., avoid the sub-quadratic increase of treewidth.

Contributions. Concretely, we provide the following.

- First, we present a novel reduction from HCF programs to SAT, which only requires linearly many auxiliary variables plus a number of auxiliary variables that is linear in the instance size and slightly superexponential in the treewidth of the SAT instance. This is achieved by guiding the whole reduction along a tree decomposition of the program. Thereby the reduction only slightly increases the treewidth, i.e., the treewidth of the resulting SAT formula is slightly larger than the treewidth of the given program.
- Then, we show that certainly we cannot avoid this increase in the treewidth. Concretely, we establish that under the widely believed *Exponential Time Hypothesis (ETH)*, one cannot decide ASP in time $2^{o(k \cdot \log(k))} \cdot n$, with treewidth k and program size n. This is in contrast to the runtime for deciding SAT: $2^{\mathcal{O}(k)} \cdot n$ with treewidth k and size n of the formula. As a result, this establishes that the consistency of normal ASP programs is already harder than SAT using treewidth. Note that this is surprising as both problems are of similar hardness according to classical complexity (NP-complete). Further, compared to known results restricting to, e.g., modular reductions (Janhunen 2006), or involving the need of auxiliary variables (Lifschitz and Razborov 2006), this shows that under ETH the increase of treewidth is indeed unavoidable when considering consistency.

Related Work. For disjunctive ASP and extensions thereof, algorithms have been proposed (Jakl, Pichler, and Woltran 2009; Pichler et al. 2014; Fichte et al. 2017) running in time linear in the instance size, but double exponential in the treewidth. Under ETH, one cannot significantly improve this runtime, using a result (Lampis and Mitsou 2017) for QBFs with quantifier depth two and a standard reduction (Eiter and Gottlob 1995) from this QBF fragment to disjunctive ASP. Unsurprisingly, SAT only requires single exponential runtime (Samer and Szeider 2010) in the treewidth. However, for normal and HCF programs only a slightly superexponential algorithm (Fichte and Hecher 2019) for solving consistency is known so far. Still, the question whether the slightly superexponentiality can be avoided was left open. The proposed algorithm was used for counting answer sets involving projection (Gebser, Kaufmann, and Schaub 2009), which is at least double exponential (Fichte et al. 2018) in the treewidth.

2 Preliminaries

Answer Set Programming (ASP). We assume familiarity with propositional satisfiability (SAT) (Biere et al. 2009; Kleine Büning and Lettman 1999), and 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 $\ell \leq m \leq n$, $a_1, \ldots, a_n$ 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 *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 for every rule $r \in \Pi$ two atoms $a \in B_r^+$ and $b \in H_r$ are joined by an edge (a, b). A head-cycle of D_Π is an $\{a, b\}$-cycle[2] for two distinct atoms a, $b \in H_r$ for some rule $r \in \Pi$. Program Π is *head-cycle-free* if D_Π contains no head-cycle (Ben-Eliyahu and Dechter 1994).

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 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 (Bidoít and Froidevaux 1991; Marek and Truszczyński 1991). A head-cycle-free program Π can be translated into a normal program in polynomial time (Ben-Eliyahu and Dechter 1994). The following characterization of answer sets is often invoked when considering normal programs (Lin and Zhao 2003). Given a set $A \subseteq \mathrm{at}(\Pi)$ of atoms. Then, a function $\varphi : A \to \{0, \ldots, |A| - 1\}$ is an *ordering* over A. Given a model I of a normal program Π and an ordering φ 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. This characterization vacuously extends to head-cycle-free programs by results of Ben-Eliyahu and Dechter (1994).

Example 1. *Consider the following program* $\Pi :=$

[2] Let $G = (V, E)$ be a digraph and $W \subseteq V$. Then, a cycle in G is a W-cycle if it contains all vertices from W.

$\{\overbrace{a \vee b \leftarrow}^{r_1}; \overbrace{c \vee e \leftarrow d}^{r_2}; \overbrace{d \vee e \leftarrow b}^{r_3}; \overbrace{b \leftarrow e, \neg d}^{r_4}; \overbrace{d \leftarrow \neg b}^{r_5}\}$.
Observe that Π *is head-cycle-free. Then,* $I := \{b, c, d\}$ *is an answer set of* Π*, since* $I \models \Pi$*, and we can prove with ordering* $\varphi := \{b \mapsto 0, d \mapsto 1, c \mapsto 2\}$ *atom* b *by rule* r_1*, atom* d *by rule* r_3*, and atom* c *by rule* r_2*. Further answer sets are* $\{b, e\}$*,* $\{a, c, d\}$*, and* $\{a, d, e\}$.

Tree Decompositions (TDs). We assume familiarity with graph terminology, cf., (Diestel 2012). 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. The *bags* $\chi_{\leq t}$ *below* t consists of the union of all bags of nodes below t in T, including 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. TDs can be 5-approximated in *single exponential time* (Bodlaender et al. 2016) in the treewidth. For a node t of T, we say that $\mathrm{type}(t)$ is *leaf* if t has no children and $\chi(t) = \emptyset$; *join* if t has children t' and t'' with $t' \neq t''$ and $\chi(t) = \chi(t') = \chi(t'')$; *int* ("introduce") if t has a single child t', $\chi(t') \subseteq \chi(t)$ and $|\chi(t)| = |\chi(t')| + 1$; *forget* if t has a single child t', $\chi(t') \supseteq \chi(t)$ and $|\chi(t')| = |\chi(t)| + 1$. If for every node t of T, $\mathrm{type}(t) \in \{$*leaf*, *join*, *int*, *forget*$\}$, the TD is called *nice*. A TD can be turned into a nice TD (Kloks 1994)[Lem. 13.1.3] *without increasing the width* in linear time.

Example 2. *Figure 1 illustrates a graph* G *and a TD* $\mathcal{T}$ *of* G *of width* 2*, which is also the treewidth of* G*, since* G *contains (Kloks 1994) a complete graph among vertices* e*,*b*,*d.

Dynamic Programming on TDs. Solvers based on *dynamic programming (DP)* evaluate a given input instance $\mathcal{I}$ in parts along a given TD of a graph representation G of the instance. Thereby, for each node t of the TD, intermediate results are stored in a *table* τ_t. This is achieved by running a *table algorithm*, which is designed for G, and stores in τ_t results of problem parts of $\mathcal{I}$, thereby considering tables $\tau_{t'}$ for child nodes t' of t. DP works for *many problems* as follows.

1. Construct a *graph representation* G of $\mathcal{I}$.
2. Compute a TD $\mathcal{T} = (T, \chi)$ of G, which is obtainable via heuristics, e.g., (Abseher, Musliu, and Woltran 2017).
3. Traverse the nodes of T in post-order (bottom-up tree traversal of T). At every node t of T during post-order traversal, execute a table algorithm that takes as input a bag $\chi(t)$, a certain *bag instance* $\mathcal{I}_t$ depending on the problem, as well as previously computed child tables of t. Then, the results of this execution is stored in table τ_t.
4. Finally, interpret table τ_n for the root node n of T in order to *output the solution* to the problem for instance $\mathcal{I}$.

In order to use TDs for ASP, we need dedicated graph representations of programs (Jakl, Pichler, and Woltran 2009). The *primal graph*[3] $\mathcal{G}_\Pi$ of program Π has the atoms of Π as vertices and an edge $\{a, b\}$ if there exists a rule $r \in \Pi$ and $a, b \in \mathrm{at}(r)$. Let $\mathcal{T} = (T, \chi)$ be a TD of primal graph $\mathcal{G}_\Pi$ of a program Π, and let t be a node of T. The *bag program* Π_t contains rules entirely covered by the bag $\chi(t)$. Formally, $\Pi_t := \{r \mid r \in \Pi, \mathrm{at}(r) \subseteq \chi(t)\}$.

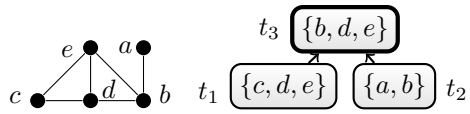

Figure 1: Graph G (left) and a tree decomposition $\mathcal{T}$ of G (right).

Example 3. *Recall program* Π *from Example 1. Observe that graph* G *of Figure 1 is the primal graph of* Π*. Further, we have* $\Pi_{t_1} = \{r_2\}$*,* $\Pi_{t_2} = \{r_1\}$*, and* $\Pi_{t_3} = \{r_3, r_4, r_5\}$*. Note that in general a rule might appear in several bag programs.*

Now, the missing ingredient for solving problems via dynamic programming along a given TD, is a suitable table algorithm. Such algorithms have been already presented for SAT (Samer and Szeider 2010) and ASP (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017; Fichte and Hecher 2019). We only briefly sketch the ideas of a table algorithm using the primal graph that computes models of a given program Π. Each table τ_t consists of rows storing interpretations over atoms in the bag $\chi(t)$. Then, the table τ_t for a leaf node t consist of the empty interpretation. For a node t with introduced variable $a \in \chi(t)$, we store in τ_t interpretations of the child table, but for each such interpretation we decide whether a is in the interpretation or not, and ensure that Π_t is satisfied. When an atom b is forgotten in a forget node t, we store interpretations of the child table, but projected to $\chi(t)$. By the properties of a TD, it is then guaranteed that all rules containing b have been processed so far. For a join node t, we store in τ_t interpretations that are in both child tables of t.

3 Treewidth-aware Reductions to SAT

Having the basic concept of dynamic programming in mind, we use this idea to design a reduction of a HCF program Π to a SAT formula F, which is treewidth-aware. The reduction is inspired by ideas of a DP algorithm for consistency of HCF programs (Fichte and Hecher 2019) and the idea of level mappings (Janhunen 2006). Intuitively, *global* orderings can cause already huge blowup in the treewidth, e.g., reductions, where all atoms are ordered at once, often cause long rules with more than treewidth many atoms. As a result, we apply these numbers only locally within the bags of a TD. More concretely, our reduction is *guided* by a TD $\mathcal{T} = (T, \chi)$ of primal graph $\mathcal{G}_\Pi$ and uses core ideas of dynamic programming along TD $\mathcal{T}$ to ensure only a slight increase in treewidth of the resulting SAT formula. Intuitively, thereby the aforementioned reduction takes care to keep the increase of width local, i.e., the increase of width happens *within* the bags of $\mathcal{T}$. Concretely, if $\mathrm{width}(\mathcal{T})$ is bounded by some value $\mathcal{O}(k)$, the treewidth of the resulting formula F is at most $\mathcal{O}(k \cdot \log(k))$.

For encoding orderings along a TD, we need the following notation. Given a TD $\mathcal{T} = (T, \chi)$ of $\mathcal{G}_\Pi$, and a node t of T. We refer to an ordering over $\chi(t)$ by t*-local ordering*.

[3] Analogously, the primal graph $\mathcal{G}_F$ of a propositional 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.

Definition 1. *A $\mathcal{T}$-*local ordering *is a set containing one t-local ordering φ_t for every t of T such that there is an interpretation I with (1)* satisfiability*: $I \models \Pi_t$ for every node t of T, (2)* provability*: for every $a \in I$, there is a node t of T and a rule $r \in \Pi_t$ proving a, and (3)* compatibility*: for every nodes t, t' of T and every $a, b \in \chi(t) \cap \chi(t')$, whenever $\varphi_t(a) < \varphi_t(b)$ then $\varphi_{t'}(a) < \varphi_{t'}(b)$.*

For an ordering φ, we use the *canonical t-local ordering* $\hat{\varphi}_t$ for each t of T as follows. Intuitively, atoms $a \in \chi(t)$ with smallest ordering position $\varphi(a)$ among all atoms in $\chi(t)$ get $\hat{\varphi}_t(a) = 0$, second-smallest get value 1, and so on. Formally, we define $\hat{\varphi}_t(a) := \mathrm{ord}_t(a, \varphi) - 1$ for each $a \in \chi(t)$, where $\mathrm{ord}_t(a, \varphi)$ is the ordinal number (rank) of a according to smallest ordering position $\varphi(a)$ among $\chi(t)$.

Example 4. *Consider program Π, answer set $I = \{b, c, d\}$, and ordering $\varphi = \{b \mapsto 0, d \mapsto 1, c \mapsto 2\}$ of Example 1. Ordering φ can easily be extended to ordering $\varphi' := \{a \mapsto 0, e \mapsto 0, b \mapsto 0, d \mapsto 1, c \mapsto 2\}$ over* at(Π)*. Then, using TD $\mathcal{T}$ of $\mathcal{G}_\Pi$, we can construct $\mathcal{T}$-local ordering $\mathcal{M} := \{\hat{\varphi}_{t_1}, \hat{\varphi}_{t_2}, \hat{\varphi}_{t_3}\}$ of φ', where $\hat{\varphi}_{t_1} = \{e \mapsto 0, d \mapsto 1, c \mapsto 2\}$, $\hat{\varphi}_{t_2} = \{a \mapsto 0, b \mapsto 0\}$, and $\hat{\varphi}_{t_3} = \{e \mapsto 0, b \mapsto 0, d \mapsto 1\}$. Consider a TD $\mathcal{T}'$ of $\mathcal{G}_\Pi$, which is similar to $\mathcal{T}$, but t_1 has a child node t', whose bag is $\{c, e\}$. Then, $\mathcal{M} \cup \{\hat{\varphi}_{t'}\}$ with $\hat{\varphi}_{t'} = \{e \mapsto 0, c \mapsto 1\}$ is a $\mathcal{T}'$-local ordering.*

In our reduction, we use the following propositional variables. For each atom $x \in \mathrm{at}(\Pi)$, we use x also as propositional variable. For each atom $x \in \chi(t)$ of each node t of T, we use $\lceil \log(|\chi(t)|) \rceil$ many variables of the form $b^i_{x_t}$ forming the i-th bit of the t-local ordering position (in binary) of x. By the shortcut notation $[\![x]\!]_{t,j}$, we refer to the *conjunction of literals over bits* $b^i_{x_t}$ for $1 \leq i \leq \lceil \log(|\chi(t)|) \rceil$ according to the representation of the number j in binary. For atoms $x, x' \in \chi(t)$ of node t of T, we use the following notation to indicate that atom x is ordered before atom x':

$$x \prec_t x' := \bigvee_{1 \leq i \leq \lceil \log(|\chi(t)|) \rceil} (b^i_{x'_t} \wedge \neg b^i_{x_t} \wedge \bigwedge_{i < j \leq \lceil \log(|\chi(t)|) \rceil} (b^j_{x_t} \longrightarrow b^j_{x'_t})).$$

Example 5. *Consider Example 4 and the $\mathcal{T}$-local ordering $\mathcal{M} = \{\varphi_{t_1}, \varphi_{t_2}, \varphi_{t_3}\}$. One could encode ordering position $\varphi_{t_1}(e) = 0$ using two bit variables $b^1_{e_{t_1}}, b^2_{e_{t_1}}$ and forcing it to false. This results in formula $[\![e]\!]_{t_1,0} = \neg b^1_{e_{t_1}} \wedge \neg b^0_{e_{t_1}}$. Then, we formulate $\varphi_{t_1}(d) = 1$ by $[\![d]\!]_{t_1,1} = \neg b^1_{d_{t_1}} \wedge b^0_{d_{t_1}}$, and $\varphi_{t_1}(c) = 2$ by $[\![c]\!]_{t_1,2} = b^1_{c_{t_1}} \wedge \neg b^0_{c_{t_1}}$. For the whole resulting formula, $e \prec_{t_1} d$, $d \prec_{t_1} c$ as well as $e \prec_{t_1} c$ hold.*

Reduction for Consistency guided by a TD. For solving consistency, we require to construct the following Formulas (1)–(6) below *for each TD node t* of T having child nodes $\mathrm{chldr}(t) = \{t_1, \ldots, t_\ell\}$. Thereby, these formulas aim at constructing $\mathcal{T}$-local orderings along the TD $\mathcal{T}$, where Formulas (1) ensure satisfiability, Formulas (2) take care of compatibility along the TD, and Formulas (6) enforce provability within a node, which is then guided along the TD by Formulas (3) to (5).

Concretely, Formulas (1) ensure that the variables of the constructed SAT formula F are such that all (bag) rules are satisfied. Then, whenever in node t an atom x has a smaller ordering position than an atom x' (using $\prec_t$), this must hold also for the parent node of t and vice versa, cf., Formulas (2). Formulas (3) guarantee, for nodes t removing bag atom x, i.e., $x \in \chi(t) \setminus \chi(t')$, that x is proven if x is set to true. Similarly, this is required for atoms $x \in \chi(n)$ that are in the root node $n = \mathrm{root}(T)$ and therefore never forgotten, cf., Formulas (4). At the same time we ensure by Formulas (5) that an atom x is proven up to node t if and only if it is proven up to some child node of t or freshly proven in node t. Finally, Formulas (6) take care that an atom x is freshly proven in node t if and only if there is at least one rule $r \in \Pi_t$ proving x.

$$\bigvee_{a \in B^+_r} \neg a \vee \bigvee_{a \in B^-_r \cup H_r} a \qquad \text{for each } r \in \Pi_t \quad (1)$$

$$(x \prec_{t'} x') \longleftrightarrow (x \prec_t x') \qquad \text{for each } t' \in \mathrm{chldr}(t), x, x' \in \chi(t) \cap \chi(t') \quad (2)$$

$$x \longrightarrow p^x_{<t'} \qquad \text{for each } t' \in \mathrm{chldr}(t), x \in \chi(t') \setminus \chi(t) \quad (3)$$

$$x \longrightarrow p^x_{<n} \qquad \text{for each } x \in \chi(n), n = \mathrm{root}(T) \quad (4)$$

$$p^x_{<t} \longleftrightarrow p^x_t \vee \Big(\bigvee_{t' \in \mathrm{chldr}(t), x \in \chi(t')} p^x_{<t'} \Big) \qquad \text{for each } x \in \chi(t) \quad (5)$$

$$p^x_t \longleftrightarrow \bigvee_{r \in \Pi_t, x \in H_r} \Big(\bigwedge_{a \in B^+_r} a \wedge x \wedge (a \prec_t x) \wedge \bigwedge_{b \in B^-_r \cup (H_r \setminus \{x\})} \neg b \Big) \qquad \text{for each } x \in \chi(t) \quad (6)$$

Example 6. *Recall program Π from Example 1, and TD $\mathcal{T}$ of $\mathcal{G}_\Pi$ given in Figure 1. We briefly show Formula F for node t_3.*

Formulas	*Formula F*
(1)	$\neg b \vee d \vee e$; $\neg e \vee d \vee b$; $d \vee b$
(2)	$(d \prec_{t_1} e) \leftrightarrow (d \prec_{t_3} e)$; $(e \prec_{t_1} d) \leftrightarrow (e \prec_{t_3} d)$
(3)	$c \rightarrow p^c_{<t_1}$; $a \rightarrow p^a_{<t_2}$
(4)	$b \rightarrow p^b_{<t_3}$; $d \rightarrow p^d_{<t_3}$; $e \rightarrow p^e_{<t_3}$
(5)	$p^b_{<t_3} \leftrightarrow (p^b_{t_3} \vee p^b_{<t_2})$;
	$p^d_{<t_3} \leftrightarrow (p^d_{t_3} \vee p^d_{<t_1})$; $p^e_{<t_3} \leftrightarrow (p^e_{t_3} \vee p^e_{<t_1})$
(6)	$p^b_{t_3} \leftrightarrow [e \wedge b \wedge (e \prec_{t_3} b) \wedge \neg d]$;
	$p^d_{t_3} \leftrightarrow [(b \wedge d \wedge (b \prec_{t_3} d) \wedge \neg e) \vee (d \wedge \neg b)]$;
	$p^e_{t_3} \leftrightarrow [b \wedge e \wedge (b \prec_{t_3} e) \wedge \neg d]$

Next, we show that the reduction is indeed aware of the treewidth and that the treewidth is only slightly increased.

Theorem 1 (Treewidth-Awareness)**.** *The reduction from a HCF program Π and a nice TD $\mathcal{T} = (T, \chi)$ of $\mathcal{G}_\Pi$ to SAT formula F consisting of Formulas (1) to (6) only slightly increases treewidth. Concretely, if k is the width of $\mathcal{T}$, then the treewidth of $\mathcal{G}_F$ is at most $\mathcal{O}(k \cdot \log(k))$.*

Proof. We construct a TD $\mathcal{T}' = (T, \chi')$ of $\mathcal{G}_F$ 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 $B(t, x) := \{b^j_{x_t} \mid x \in \chi(t), 1 \leq j \leq \lceil \log(|\chi(t)|) \rceil\}$. We inductively define $\chi'(t) := \chi(t) \cup (\bigcup_{x \in \chi(t)} B(t, x) \cup B(\hat{t}, x)) \cup \{p^y_{<t'}, p^x_t \mid t' \in \{t, t_1, \ldots, t_\ell\}, x \in \chi(t), y \in \chi(t')\}$. Observe that indeed $\mathcal{T}'$ is a TD of $\mathcal{G}_F$. Further, $|\chi'(t)| \leq k + k \cdot \lceil \log(k) \rceil \cdot 2 + k \cdot (\ell + 2)$. Thus, the width of nice $\mathcal{T}'$ is in $\mathcal{O}(k \cdot \log(k))$. □

Later we will see the lower bound for consistency of normal ASP, which indicates that one cannot expect to significantly improve this increase of treewidth. Next, we present consequences for auxiliary variables and runtime.

Corollary 1 (Runtime)**.** *The reduction from a HCF program Π and a nice TD $\mathcal{T}$ of $\mathcal{G}_\Pi$ to SAT formula F consisting of Formulas (1) to (6) uses at most $\mathcal{O}(k \cdot \log(k) \cdot h)$ many variables and runs in time $\mathcal{O}(k \cdot \log(k) \cdot h + |\Pi|)$, where k and h form the width and the number of nodes of $\mathcal{T}$, respectively.*

Proof. The result follows from Theorem 1. Linear time in Π can be obtained by slightly modifying Formulas (1) and (6) such that each rule $r \in \Pi$ is used in only one node t, where $r \in \Pi_{t'}$, but $r \notin \Pi_t$, for some $t' \in \mathrm{chldr}(t)$. □

Note that a nice TD of $\mathcal{G}_\Pi$ of width $k = tw(\mathcal{G}_\Pi)$, having only $h = \mathcal{O}(|\mathrm{at}(\Pi)|)$ many nodes (Kloks 1994)[Lem. 13.1.2] always exists. Further, since $k \cdot \log(k)$ might be much smaller than $\log(|\mathrm{at}(\Pi)|)$, for some programs this reduction might pay off compared to global or component-based orderings used in tools like lp2sat (Janhunen 2006) or lp2acyc (Gebser, Janhunen, and Rintanen 2014; Bomanson et al. 2016).

Correctness of the Reduction. Now, we discuss correctness of our reduction, which establishes that $\mathcal{T}$-local orderings encoded by Formulas (1) to (6) follow ideas of the characterization of answer sets for HCF programs.

Theorem 2 (Correctness)**.** *The reduction from a HCF program Π and a TD $\mathcal{T} = (T, \chi)$ of $\mathcal{G}_\Pi$ to SAT formula F consisting of Formulas (1) to (6) is correct. Concretely, for each answer set of Π there is a model of F and vice versa.*

Proof. "⇒": Given an answer set M of Π. Then, there is an ordering φ over $\mathrm{at}(\Pi)$, where every atom of M is proven. Next, we construct a model I of F as follows. For each $x \in \mathrm{at}(\Pi)$, we let (c1) $x \in I$ if $x \in M$. For each node t of T, and $x \in \chi(t)$: (c2) For every $l \in [\![x]\!]_{t,i}$ with $i = \hat{\varphi}_t(x)$, we set $l \in I$ if l is a variable. (c3) If there is a rule $r \in \Pi_t$ proving x, we let both $p^x_{<t}, p^x_t \in I$. Finally, (c4) we set $p^x_{<t} \in I$, if $p^x_{<t'} \in I$ for $t' \in \mathrm{chldr}(t)$.

It remains to show that I is indeed a model of F. By (c1), Formulas (1) are satisfied by I. Further, by (c2) of I, the order of φ is preserved among $\chi(t)$ for each node t of T, therefore Formulas (2) are satisfied by I. Further, by definition of TDs, for each rule $r \in \Pi$ there is a node t with $r \in \Pi_t$. Consequently, M is proven with ordering φ, for each $x \in M$ there is a node t and a rule $r \in \Pi_t$ proving x. Then, Formulas (6) are satisfied by I due to (c3), and Formulas (5) are satisfied by I due to (c4). Finally, by connectedness of TDs, also Formulas (3) and (4) are satisfied.

"⇐": Given any model I of F. Then, we construct an answer set M of Π as follows. We set $a \in M$ if $a \in I$ for any $a \in \mathrm{at}(\Pi)$. We define for each node t a t-local ordering φ_t, where we set $\varphi_t(x)$ to j for each $x \in \chi(t)$ such that j is the decimal number of the binary number for x in t given by I. Concretely, $\varphi_t(x) := j$, where j is such $I \models [\![x]\!]_{t,j}$. Then, we define an ordering φ iteratively as follows. We set $\varphi(a) := 0$ for each $a \in \mathrm{at}(\Pi)$, where there is no node t of T with $\varphi_t(b) < \varphi_t(a)$. Then, we set $\varphi(a) := 1$ for each $a \in \mathrm{at}(\Pi)$, where there is no node t of T with $\varphi_t(b) < \varphi_t(a)$ for some $b \in \chi(t)$ not already assigned in the previous iteration, and so on. In turn, we construct φ iteratively by assigning increasing values to φ. Observe that φ is well-defined, i.e., each atom $a \in \mathrm{at}(\Pi)$ gets a unique value since it cannot be the case for two nodes t, t' and atoms $x, x' \in \chi(t) \cap \chi(t')$ that $\varphi_t(x) < \varphi_t(x')$, but $\varphi_{t'}(x) \geq \varphi_{t'}(x')$. Indeed, this is prohibited by Formulas (2) and connectedness of $\mathcal{T}$ ensuring that $\mathcal{T}$ restricted to x is still connected.

It remains to show that φ is an ordering for Π proving M. Assume towards a contradiction that there is an atom $a \in M$ that is not proven. Observe that either a is in the bag $\chi(n)$ of the root node n of T, or it is forgotten below n. In both cases we require a node t such that $p^x_{<t} \notin I$ by Formulas (4) and (3), respectively. Consequently, by connectedness of $\mathcal{T}$ and Formulas (5) there is a node t', where $p^x_{t'} \in I$. But then, since Formulas (6) are satisfied by I, there is a rule $r \in \Pi_{t'}$ proving a with $\varphi_{t'}$. Therefore, since by construction of φ, there cannot be a node t of T with $x, x' \in \chi(t)$, $\varphi_t(x) < \varphi_t(x')$, but $\varphi(x) \geq \varphi(x')$, r is proving a with φ. □

Strengthening the SAT Formula. Next, we strengthen the previous reduction to ensure to get rid of duplicate $\mathcal{T}$-local orderings for a particular answer set of Π.

In Formulas (7), we ensure that if a variable $x \in \mathrm{at}(\Pi)$ is set to false, then its ordering position is zero. Formulas (8) make sure that if the position of x is set to $i \geq 1$ in node t, there has to be a bag atom y having position $i-1$. Intuitively, if this is not the case we could shift the position of x from i to $i-1$. Finally, Formulas (9) ensure that whenever in a node t there is a rule $r \in \Pi_t$ with $x \in H_r$ and x has position $i \geq 1$, either there is at least one atom $y \in B^+_r$ having position $i-1$, or r is not proving x.

$$\neg x \longrightarrow \bigwedge_{1 \leq j \leq \lceil \log(|\chi(t)|) \rceil} \neg b^j_{x_t} \quad \text{for each } x \in \chi(t) \tag{7}$$

$$[\![x]\!]_{t,i} \longrightarrow \bigvee_{y \in \chi(t) \setminus \{x\}} [\![y]\!]_{t,i-1} \quad \text{for each } x \in \chi(t),\ 1 \leq i < |\chi(t)| \tag{8}$$

$$\bigwedge_{r \in \Pi_t, x \in H_r, 1 \leq i < |\chi(t)|} ([\![x]\!]_{t,i} \longrightarrow \bigvee_{a \in B^+_r} \neg a \vee (a \not<_t x) \vee \bigvee_{b \in B^-_r \cup (H_r \setminus \{x\})} b \vee \bigvee_{y \in B^+_r} [\![y]\!]_{t,i-1}) \quad \text{for each } x \in \chi(t) \tag{9}$$

In general, we do not expect to get rid of all redundant $\mathcal{T}$-local orderings for an answer set, though. The reason for this expectation lies in the fact that the different (chains of) rules required for setting the position for an atom a might be "spread" among the whole tree decomposition. Consequently, one would need to compare different, *absolute* values of orderings, cf., (Janhunen 2006), instead of the ordering positions relative to one TD node as presented here, which requires to store for each atom in the worst case numbers up to $|\mathrm{at}(\Pi)|$. Obviously, this number is then not bounded by the treewidth, and one cannot encode it without increasing the treewidth in general. However, if for each answer set M of Π, and every $a \in M$, there can be only one rule $r \in \Pi$, where $a \in H_r \cap M$ and $M \cap (H_r \setminus \{a\}) = \emptyset$, that is satisfied by M, then there is a bijective correspondence between

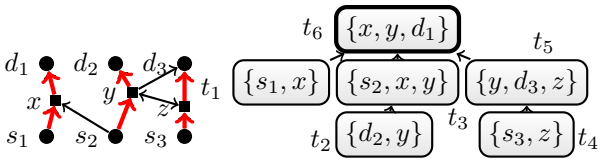

Figure 2: An instance $I = (G, P)$ (left) of the DISJOINT PATHS PROBLEM and a TD of G (right).

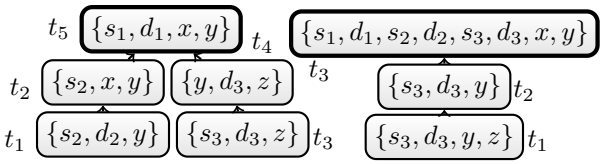

Figure 3: A pair-respecting TD (left), and a pair-connected TD $\mathcal{T}$ (right) of (G, P) of Figure 2.

answer sets of Π and models of Formulas (1) to (9). One example of such programs is constructed in the next section.

4 Why ASP Consistency is Harder than SAT

This section concerns the hardness of ASP consistency when considering treewidth. The high-level reason for ASP being harder than SAT when assuming bounded treewidth, lies in the issue that a TD, while capturing the structural dependencies of a program, might force an evaluation that is completely different from the orderings proving answer sets. Consequently, during dynamic programming for ASP, one needs to store in each table τ_t for each node t during post-order traversal, in addition to an interpretation (candidate answer set), also an ordering among the atoms in those interpretations. We show that under reasonable assumptions in complexity theory, this worst-case cannot be avoided. Then, the resulting runtime consequences cause ASP to be slightly harder than SAT, where in contrast to ASP storing a table τ_t of only assignments for each node t suffices.

We show our novel hardness result by reducing from the (DIRECTED) DISJOINT PATHS PROBLEM, which is a graph problem defined as follows. Given a directed graph $G = (V, E)$, and a set $P \subseteq V \times V$ of disjoint pairs of the form (s_i, d_i) consisting of *source* s_i and *destination* d_i, where $s_i, d_i \in V$ such that each vertex occurs at most once in P, i.e., $\left|\bigcup_{(s_i,d_i)\in P}\{s_i, d_i\}\right| = 2 \cdot |P|$. Then, (G, P) is an instance of the DISJOINT PATHS PROBLEM, asking whether there exist $|P|$ many (vertex-disjoint) paths from s_i to d_i for $1 \leq i \leq |P|$. Concretely, each vertex of G is allowed to appear in at most one of these paths. For the ease of presentation, we assume without loss of generality (Lokshtanov, Marx, and Saurabh 2011) that sources s_i have no incoming edge (x, s_i), and destinations d_i have no outgoing edge (d_i, x).

Example 7. *Figure 2 (left) shows an instance $I = (G, P)$ of the* DISJOINT PATHS PROBLEM*, where P consists of pairs of the form (s_i, d_i). The only solution to I is both emphasized and colored in red. Figure 2 (right) depicts a TD of G.*

While under ETH, SAT cannot be solved in time $2^{o(k)} \cdot \text{poly}(|\text{at}(F)|)$, where k is the treewidth of the primal graph of a given propositional formula F, the DISJOINT PATHS PROBLEM is considered to be even harder. Concretely, the problem has been shown to be slightly superexponential as stated in the following proposition.

Proposition 1 ((Lokshtanov, Marx, and Saurabh 2011))**.** *Under ETH, the* DISJOINT PATHS PROBLEM *is slightly superexponential, i.e., any instance (G, P) with $G = (V, E)$ cannot be solved in time $2^{o(k \cdot \log(k))} \cdot \text{poly}(|V|)$, where $k = tw(G)$.*

It turns out that the DISJOINT PATHS PROBLEM is a suitable problem candidate for showing the hardness of ASP. Next, we require the following notation of open pairs, whose result is then applied in our reduction. Given an instance (G, P) of the DISJOINT PATHS PROBLEM, a TD $\mathcal{T} = (T, \chi)$ of G, and a node t of T. Then, a pair $(s, d) \in P$ is *open in node t*, if either $s \in \chi_{\leq t}$ ("*open due to source s*") or $d \in \chi_{\leq t}$ ("*open due to destination d*"), but not both.

Proposition 2 ((Scheffler 1994))**.** *An instance (G, P) of the* DISJOINT PATHS PROBLEM *does not have a solution if there is a TD $\mathcal{T} = (T, \chi)$ of G and a bag $\chi(t)$ with more than $|\chi(t)|$ many pairs in P that are open in a node t of T.*

Proof. The result, cf., (Scheffler 1994), boils down to the fact that each bag $\chi(t)$, when removed from G, results in a disconnected graph consisting of two components. Between these components can be at most $|\chi(t)|$ different paths. □

Preparing pair-connected TDs. Before we present the actual reduction, we need to define a *pair-respecting* tree decomposition of an instance (G, P) of the DISJOINT PATHS PROBLEM. Intuitively, such a TD of G additionally ensures that each pair in P is encountered together in some TD bag.

Definition 2. *A TD $\mathcal{T} = (T, \chi)$ of G is a* pair-respecting TD *of (G, P) if for any pair $p = (s, d)$ with $p \in P$, (1) whenever p is open in a node t due to s, or due to d, then $s \in \chi(t)$, or $d \in \chi(t)$, respectively. Further, (2) whenever p is open in a node t, but not open in the parent t' of t ("p is* closed in t'*"), both $s, d \in \chi(t')$.*

We observe that such a pair-respecting TD can be computed with only a linear increase in the (tree)width in the worst case. Concretely, we can turn any TD $\mathcal{T} = (T, \chi)$ of G into a pair-respecting TD $\mathcal{T}' = (T, \chi')$ of (G, P). Thereby, the tree T is traversed for each t of T in post-order, and vertices of P are added to $\chi(t)$ accordingly, resulting in $\chi'(t)$, such that conditions (1) and (2) of pair-respecting TDs are met. Observe, that this doubles the sizes of the bags in the worst case, since by Proposition 2 there can be at most bag-size many open pairs.

Example 8. *Figure 3 (left) shows a pair-respecting TD of (G, P) of Figure 2, which can be obtained by transforming the TD of Figure 2 (right), followed by simplifications.*

Given a sequence σ of pairs of P in the order of closure with respect to the post-order of T. We refer to σ by the *closure sequence* of $\mathcal{T}$. We denote by $p \in_i \sigma$ that pair p is the *pair closed i-th* in the order of σ. Intuitively, e.g., the first pair $p \in_1 \sigma$ indicates that pair $p \in P$ is the first to be closed when traversing T in post-order.

Definition 3. *A* pair-connected TD *$\mathcal{T}$=(T, χ) of (G, P) is a pair-respecting TD of (G, P), if, whenever a pair $p \in_i \sigma$ with i>1 is closed in a node t of T, also for the pair $(s, d) \in_{i-1} \sigma$ closed directly before p in σ, both $s, d \in \chi(t)$.*

We can turn any pair-respecting, *nice* TD $\mathcal{T}'=(T,\chi')$ of width k into a pair-connected TD $\mathcal{T}''=(T,\chi'')$ with constant increase in the width. Let therefore pair $p \in_i \sigma$ be closed ($i>1$) in a node t, and pair $(s,d) \in_{i-1}$ be closed before p in node t'. Intuitively, we need to add s,d to all bags $\chi'(t'),\ldots,\chi'(t)$ of nodes encountered after node t' and before node t of the post-order tree traversal, resulting in χ''. However, the width of $\mathcal{T}''$ is at most $k+3\cdot|\{s,d\}|=k+6$, since in the tree traversal each node of T is passed at most 3 times, namely when traversing down, when going from the left branch to the right branch, and then also when going upwards. Indeed, to ensure $\mathcal{T}''$ is a TD (connectedness condition), we add at most 6 additional atoms to every bag.

Example 9. *Figure 3 (right) depicts a pair-connected TD of (G,P) of Figure 2, obtainable by transforming the pair-respecting TD of Figure 3 (left), followed by simplifications.*

4.1 Reducing from DISJOINT PATHS to ASP

In this section, we show the main reduction R of this paper, assuming any instance $I=(G,P)$ of the DISJOINT PATHS PROBLEM. Before we construct our program Π, we require a nice, pair-connected TD $\mathcal{T}=(T,\chi)$ of G, whose width is k and a corresponding closure sequence σ. By Proposition 2, for each node t of $\mathcal{T}$, there can be at most k many open pairs of P, which we assume in the following. If this was indeed not the case, we can immediately output, e.g., $\{a \leftarrow \neg a\}$.

Then, we use the following atoms in our reduction. Atoms $e_{u,v}$, or $ne_{u,v}$ indicate that edge $(u,v)\in E$ is used, or unused, respectively. Then, r_u for any vertex $u\in V$ indicates that u is reached via used edges, and r^*_d are auxiliary reachability atoms for destination vertices d (i.e., where $(s,d)\in P$). Finally, we also need atom f^u_t for a node t of T, and vertex $u\in\chi(t)$, to indicate that vertex u is already finished in node t, i.e., u has one used, outgoing edge. The presence of this atom f^u_t in an answer set prohibits to take additional edges of u in parent nodes of t, which is needed due to the need of disjoint paths of the DISJOINT PATHS PROBLEM.

The instance $\Pi=R(I,\mathcal{T})$ constructed by reduction R consists of three program parts, namely *reachability* $\Pi_\mathcal{R}$, *linking* $\Pi_\mathcal{L}$ of two pairs in P, as well as *checking* $\Pi_\mathcal{C}$ of disjointness of constructed paths. Consequently, $\Pi=\Pi_\mathcal{R}\cup\Pi_\mathcal{L}\cup\Pi_\mathcal{C}$. All three programs $\Pi_\mathcal{R}$, $\Pi_\mathcal{L}$, and $\Pi_\mathcal{C}$ are guided along TD $\mathcal{T}$, which ensures that the width of Π is only linearly increased. Note that this has to be carried out carefully. In particular, since the number of atoms of the form $e_{u,v}$ using only vertices u,v that appear in one bag, can be already quadratic in the bag size. The goal of this reduction, however, admits only a linear overhead in the bag size. Consequently, we are, e.g., not allowed to construct rules in Π that require more than $\mathcal{O}(k)$ edges in one bag of a TD of $\mathcal{G}_\Pi$.

To this end, let the *ready edges E^{re}_t in node t* be the set of edges $(u,v)\in E$ not present in t anymore, i.e., $\{u,v\}\subseteq\chi(t')\setminus\chi(t)$ for any child node $t'\in\mathrm{chldr}(t)$. Further, let E^{re}_n for the root node $n=\mathrm{root}(T)$ additionally contain also all edges of n, i.e., $E\cap(\chi(n)\times\chi(n))$. Intuitively, ready edges for t will be processed in node t. Note that each edge occurs in exactly one set of ready edges. Further, for nice TDs $\mathcal{T}$, we always have $|E^{\mathrm{re}}_t|\leq k$, i.e., ready edges are linear in k.

Example 10. *Recall instance $I=(G,P)$ with $G=(V,E)$ of Figure 2, and pair-connected TD $\mathcal{T}=(T,\chi)$ of I of Figure 3 (right). Then, $E^{re}_{t_1}=\emptyset$, $E^{re}_{t_2}=\{(y,z),(z,y),(z,d_3),(s_3,z)\}$, since $z\notin\chi(t_2)$, and $E^{re}_{t_3}=E\setminus E^{re}_{t_2}$ for root t_3 of $\mathcal{T}$.*

Reachability $\Pi_\mathcal{R}$. Program $\Pi_\mathcal{R}$ is constructed as follows.

$$e_{u,v}\leftarrow r_u,\neg ne_{u,v} \quad \text{for each } (u,v)\in E^{\mathrm{re}}_t \tag{10}$$
$$ne_{u,v}\leftarrow\neg e_{u,v} \quad \text{for each } (u,v)\in E^{\mathrm{re}}_t \tag{11}$$
$$r_v\leftarrow e_{u,v} \quad \text{for each } (u,v)\in E^{\mathrm{re}}_t,(s,v)\notin P \tag{12}$$
$$r^*_d\leftarrow e_{u,d} \quad \text{for each } (u,d)\in E^{\mathrm{re}}_t,(s,d)\in P \tag{13}$$

Rules (10) and (11) ensure that there is a partition of edges in used edges $e_{u,v}$ and unused edges $ne_{u,v}$. Additionally, Rules (10) take care that only edges of adjacent, reachable vertices are used. Naturally, this requires that initially at least one vertex is reachable (constructed below). Rules (12) and (13) ensure reachability r_v and r^*_v over used edges $e_{u,v}$ for non-destination vertex v and destination v, respectively.

Linking of pairs $\Pi_\mathcal{L}$. Program $\Pi_\mathcal{L}$ is constructed as follows.

$$\leftarrow\neg r_d \quad \text{for each } (s,d)\in P \tag{14}$$
$$r_{s_1}\leftarrow \quad \text{for } (s_1,d)\in_1\sigma \tag{15}$$
$$r_{s_i}\leftarrow r_{d_{i-1}} \quad \text{for each } (s_i,d)\in_i\sigma,(s,d_{i-1})\in_{i-1}\sigma \tag{16}$$
$$r_{d_1}\leftarrow r^*_{d_1} \quad \text{for } (s,d_1)\in_1\sigma \tag{17}$$
$$r_{d_i}\leftarrow r^*_{d_i},r_{d_{i-1}} \quad \text{for each } (s,d_i)\in_i\sigma,(s',d_{i-1})\in_{i-1}\sigma \tag{18}$$

Rules (14) make sure that, ultimately, destination vertices of all pairs are reached. As an initial, reachable vertex, Rule (15) sets the source vertex s reachable, whose pair is closed first. Then, the linking of pairs is carried out along the TD in the order of closure, as given by σ. Thereby, Rules (16) conceptually construct auxiliary links (similar to edges) between different pairs, in the order of σ, which is guided along the TD to ensure only a linear increase in treewidth of $\mathcal{G}_\Pi$ of the resulting program Π. Interestingly, these additional dependencies, since guided along the TD, do not increase the treewidth by much as we will see in the next subsection. Rule (17) makes sure that if destination vertex d_1 of the pair closed first is auxiliary-reached ($r^*_{d_1}$), reachability r_{d_1} is set.

Then, it is *crucial* that we prevent a source vertex s_i of a pair $(s_i,d_i)\in_i\sigma$ from reaching a destination vertex d_j of a pair $(s_j,d_j)\in_j\sigma$ preceding (s_i,d_i) in σ, i.e., $j<i$. To this end, we need to construct parts of cycles that prevent this. Concretely, if some source s_i reaches to d_j, i.e., d_j is reachable via s_i, the goal is to have a cyclic reachability from d_j to s_i, with no external support for corresponding reachability atoms. Actually, Rules (16) also have the purpose of aiding in construction of these potential positive cycles. Together with Rules (18) we achieve that if d_j is reachable, this cannot be due to s_i, since reachability of $d_{i-1},d_{i-2},\ldots,d_j$ (therefore s_i itself) is required for reachability of s_i. Consequently, assuming that there is no external support for these reachability atoms (which we will ensure in program $\Pi_\mathcal{C}$ below), and that if s_i is reachable, d_j is reachable, we end up with cyclic reachability without external support. Figure 4 shows the positive dependency graph $D_{R_\mathcal{L}}$ of Rules (16)–(18), where pairs $(s_i,d_i)\in_i\sigma$, discussed in the following example.

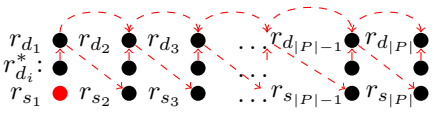

Figure 4: Positive dependency graph $D_{R_\mathcal{L}}$ of Rules (16)–(18) constructed for any closure sequence σ such that $(s_i, d_i) \in_i \sigma$.

Example 11. *Consider the dependency graph $D_{R_\mathcal{L}}$ of Rules (16) and (18), as depicted in Figure 4. Observe that whenever s_i reaches some d_j with $j < i$, this causes a cycle $C{=}r_{s_i}, \ldots, r_{d_j}, r_{d_{j+1}}, \ldots, r_{d_{i-1}}, r_{s_i}$ over reachability atoms (cyclic dependency). If each vertex u of G can have at most one outgoing edge, i.e., only one atom $e_{u,v}$ in an answer set of $\Pi = R(I, \mathcal{T})$, no atom of C can be proven (no external support). Note that C could also be constructed by adding $\mathcal{O}(|P|^2)$ many edges from d_i to d_j for $j > i$. However, this would cause an increase of structural dependency for d_i, and in fact, the treewidth increase would be beyond linear.*

Checking of disjointness $\Pi_\mathcal{C}$. Finally, we create rules in Π that enforce at most one outgoing, used edge per vertex. This is required to ensure that we do not use a vertex twice, as required by the DISJOINT PATHS PROBLEM. We do this by guiding the information, whether the corresponding outgoing edge was used, via atoms f^u_t along the TD to ensure that the treewidth is not increased significantly. Having at most one outgoing, used edge per vertex of G further ensures that when a source of a pair p reaches a destination of a pair preceding p in σ, then no atom of the resulting cycle as constructed in $\Pi_\mathcal{L}$ will be provable (no external support). Consequently, in the end every source of p has to reach the destination of p by the pigeon hole principle. Program $\Pi_\mathcal{C}$ is constructed for every node t with $t', t'' \in \mathrm{chldr}(t)$, if t has child nodes, as follows.

$$f^u_t \leftarrow e_{u,v} \qquad \text{for each } (u,v) \in E^{\mathrm{re}}_t, u \in \chi(t) \qquad (19)$$

$$f^u_t \leftarrow f^u_{t'} \qquad \text{for each } u \in \chi(t) \cap \chi(t') \qquad (20)$$

$$\leftarrow f^u_{t'}, f^u_{t''} \qquad \text{for each } u \in \chi(t') \cap \chi(t''), t' \neq t'' \qquad (21)$$

$$\leftarrow f^u_{t'}, e_{u,v} \qquad \text{for each } (u,v) \in E^{\mathrm{re}}_t, u \in \chi(t') \qquad (22)$$

$$\leftarrow e_{u,v}, e_{u,w} \qquad \text{for each } (u,v),(u,w) \in E^{\mathrm{re}}_t,\ v{\neq}w \qquad (23)$$

Rules (19) ensure that the finished flag f^u_t is set for used edges $e_{u,v}$. Then, this information of $f^u_{t'}$ is guided along the TD from child node t' to parent node t by Rules (20). If for a vertex $u \in V$ we have $f^u_{t'}$ and $f^u_{t''}$ for two different child nodes $t', t'' \in \mathrm{chldr}(t)$, this indicates that two different edges were encountered both below t' and below t''. Consequently, this situation is avoided by Rules (21). Rules (22) make sure to disallow additional edges for vertex u in a TD node t, if the flag $f^u_{t'}$ of child node t' is set. Finally, Rules (23) prohibit two different edges for the same vertex u within a TD node.

Example 12. *Recall instance $I = (G, P)$ with $G = (V, E)$ of Figure 2, pair-connected TD $\mathcal{T} = (T, \chi)$ of I of Figure 3 (right), and $E^{re}_{t_2} = \{(y,z),(z,y),(z,d_3),(s_3,z)\}$. We briefly present the construction of $\Pi_\mathcal{C}$ for node t_2.*

Rules	$\Pi_\mathcal{L}$
(19)	$f^y_{t_2} \leftarrow e_{y,z}$; $f^{s_3}_{t_2} \leftarrow e_{s_3,z}$
(20)	$f^{s_3}_{t_2} \leftarrow f^{s_3}_{t_1}$; $f^{d_3}_{t_2} \leftarrow f^{d_3}_{t_1}$; $f^{y}_{t_2} \leftarrow f^{y}_{t_1}$
(22)	$\leftarrow f^y_{t_1}, e_{y,z}$; $\leftarrow f^z_{t_1}, e_{z,y}$; $\leftarrow f^z_{t_1}, e_{z,d_3}$; $\leftarrow f^{s_3}_{t_1}, e_{s_3,z}$
(23)	$\leftarrow e_{z,y}, e_{z,d_3}$

4.2 Correctness and Runtime Analysis

First, we show that the reduction is indeed correct, followed by a result stating that the treewidth of the reduction is at most linearly worsened, which is crucial for the runtime lower bound to hold. Then, we present the runtime and the (combined) main result of this paper.

Lemma 1 (≤ 1 Outgoing Edge)**.** *Given any instance $I = (G, P)$ of the* DISJOINT PATHS PROBLEM*, and any answer set M of $R(I, \mathcal{T})$ using any pair-connected TD $\mathcal{T}$ of (G, P). Then, there cannot be two edges of the form $e_{u,v}, e_{u,w} \in M$.*

Proof. Assume towards a contradiction that there are three different vertices $u, v, w \in V$ with $e_{u,v}, e_{u,w} \in M$. Then, by Rules (23) there cannot be a node t with $(u,v),(u,w) \in E^{\mathrm{re}}_t$. However, by the definition of TDs, there are nodes t', t'' with $(u,v) \in E^{\mathrm{re}}_{t'}$ and $(u,w) \in E^{\mathrm{re}}_{t''}$. By connectedness of TDs, u appears in each bag of any node of the path X between t' and t''. Then, either t' is an ancestor of t'' (or vice versa, symmetrical) or there is a common ancestor t. In the former case, $f^u_{t''}$ is justified by Rules (19) and so is $f^u_{\hat{t}}$ on each node $\hat{t}$ of X by Rules (20) and therefore ultimately Rules (22) fail due to $f^u_{t'}, e_{u,w} \in M$. In the latter case, $f^u_{t''}, f^u_{t'}$ is justified by Rules (19) and so is $f^u_{\hat{t}}$ on each node $\hat{t}$ of X by Rules (20). Then, Rules (21) fail due to $f^u_{t'}, f^u_{t''} \in M$. □

Theorem 3 (Correctness)**.** *Reduction R as proposed in this section is correct. More concretely, given an instance $I = (G, P)$ of the* DISJOINT PATHS PROBLEM*, and a pair-connected TD $\mathcal{T} = (T, \chi)$ of G. Then, I has a solution if and only if the program $R(I, \mathcal{T})$ admits an answer set.*

Proof. "⇒": Given any positive instance I of DISJOINT PATHS PROBLEM. Then, there are disjoint paths $P_1, \ldots, P_i, \ldots P_{|P|}$ from s_1 to d_1, …, s_i to d_i, …, $s_{|P|}$ to $d_{|P|}$ for each pair $(s_i, d_i) \in P$. Assuming further pair-connected TD $\mathcal{T}$ of I, we construct in the following an answer set M of $\Pi = R(I, \mathcal{T})$. To this end, we collect reachable atoms $A := \{u \mid u \text{ appears in some } P_i, 1 \leq i \leq |P|\}$ and used edges $U := \{(u,v) \mid v \text{ appears immediately after } u \text{ in some } P_i, 1 \leq i \leq |P|\}$. Then, we construct answer set candidate $M := \{r^*_{d_i} \mid 1 \leq i \leq |P|\} \cup \{r_u \mid u \in A\} \cup \{e_{u,v} \mid (u,v) \in U\} \cup \{ne_{u,v} \mid (u,v) \in E \setminus U\} \cup \{f^u_t \mid (u,v) \in U \cap E^{\mathrm{re}}_t\} \cup \{f^u_t \mid (u,v) \in U \cap E^{\mathrm{re}}_{t'}, u \in \chi(t), t' \text{ is a descendant of } t \text{ in } T\}$. It remains to show that M is an answer set of Π. Observe that M indeed satisfies all the rules of $\Pi_\mathcal{R}$. In particular, by construction, we have reachability r_v for every vertex v of every pair in P, and the partition in used edges $e_{u,v}$ and unused edges $ne_{u,v}$ is ensured. Further, $\Pi_\mathcal{L}$ is satisfied, as, again by construction, for each vertex v of every pair in P, we have $r_v \in M$. Finally, $\Pi_\mathcal{C}$ is satisfied as by construction $f^u_t \in M$ iff $e_{u,v} \in M \cap E^{\mathrm{re}}_t$ or $e_{u,v} \in M \cap E^{\mathrm{re}}_{t'}$ for any descendant node t' of t with $u \in \chi(t)$. It is easy to see that M is indeed a $\subseteq$-smallest model of the reduct Π^M, since, atoms for used and unused edges form a partition of E.

"⇐": Given any answer set M of Π. First, we observe that we can only build paths from sources towards destinations, as sources have only outgoing edges and destinations allow only incoming edges. Further, by construction, vertices can

only have one used, outgoing edge, cf., Lemma 1. Consequently, if a vertex had more than one used, incoming edge, one cannot match at least one pair of P (by combinatorial pigeon hole principle). Hence, in an answer set M of Π, there is at most one incoming edge per vertex. By construction of Π, in order to reach each d_i with $(s_i, d_i) \in_i \sigma$, s_i cannot reach some $d_{j'}$ with $j' < i$. Towards a contradiction assume otherwise, i.e., s_i reaches $d_{j'}$. But then, by construction of the reduction, we also have a reachable path from $d_{j'}$ to s_i, consisting of $d_{j'}, d_{j'+1}, \ldots, d_{i-1}, s_i$. Since every vertex has at most one incoming edge, $d_{j'}$ cannot have any other justification for being reachable, nor does any source on this path. Hence, this forms a cycle without external support, which can not be present in an answer set. Therefore, s_i only reaches d_i, since otherwise there would be at least one vertex s_j required to reach $s_{i'}$ with $(s_{i'}, d_{i'}) \in_{i'} \sigma$, $i' < j$. Consequently, we construct a witnessing path P_i for each pair $(s, d) \in_i \sigma$ as follows: $P_i := s, p_1, \ldots, p_m, d$ where $\{e_{s,p_1}, e_{p_1,p_2}, \ldots, e_{p_{m-1},p_m}, e_{p_m,d}\} \subseteq M$. Thus, P_i starts with s, follows used edges in M and reaches d. □

Lemma 2 (Treewidth-Awareness)**.** *Given an instance $I = (G, P)$ of the* DISJOINT PATHS PROBLEM*, and a pair-connected, nice TD $\mathcal{T}$ of I of width k. Then, the treewidth of $\mathcal{G}_\Pi$, where $\Pi = R(I, \mathcal{T})$ is obtained by R, is at most $\mathcal{O}(k)$.*

Proof. Given any pair-connected, nice TD $\mathcal{T} = (T, \chi)$ of $I = (G, P)$. Since $\mathcal{T}$ is nice, a node in T has at most $\ell = 2$ many child nodes. From $\mathcal{T}$ we construct a TD $\mathcal{T}' = (T, \chi')$ of $\mathcal{G}_\Pi$. Thereby we set for every node t of T, $\chi'(t) := \{r_u, f_t^u \mid u \in \chi(t)\} \cup \{r_d^* \mid d \in \chi(t), (s, d) \in P\} \cup \{e_{u,v}, ne_{u,v}, r_u, r_v, f_{t'}^u \mid (u, v) \in E_t^{\text{re}}, t' \in \text{chldr}(t), u \in \chi(t')\} \cup \{f_{t'}^u, f_t^u \mid t' \in \text{chldr}(t), u \in \chi(t) \cap \chi(t')\}$. Observe that $\mathcal{T}'$ is a valid TD of $\mathcal{G}_\Pi$. Further, by construction we have $|\chi'(t)| \leq 2 \cdot |\chi(t)| + |\chi(t)| + (4+\ell) \cdot k + (\ell+1) \cdot |\chi(t)|$, since $|E_t^{\text{re}}| \leq k$. The claim sustains for nice TDs ($\ell = 2$). □

Corollary 2 (Runtime)**.** *Reduction R as proposed in this section runs for a given instance $I = (G, P)$ of the* DISJOINT PATHS PROBLEM *with $G = (V, E)$, and a pair-connected, nice TD $\mathcal{T}$ of I of width k and h many nodes, in time $\mathcal{O}(k \cdot h)$.*

Next, we are in the position of showing the main result, namely the normal ASP lower bound.

Theorem 4 (Lower bound)**.** *Given an arbitrary normal or HCF program Π, where k is the treewidth of the primal graph of Π. Then, unless ETH fails, the consistency problem for Π cannot be solved in time $2^{o(k \cdot \log(k))} \cdot \text{poly}(|\text{at}(\Pi)|)$.*

Proof. Let (G, P) be an instance of the DISJOINT PATHS PROBLEM. First, we construct (Bodlaender et al. 2016) a nice TD $\mathcal{T}$ of $G = (V, E)$ of treewidth k in time $c^k \cdot |V|$ for some constant c such that the width of $\mathcal{T}$ is at most $5k + 4$. Then, we turn the result into a pair-connected TD $\mathcal{T}' = (T', \chi')$, thereby having width at most $k' = 2 \cdot (5k + 4) + 6$. Then, we construct program $\Pi = R(I, \mathcal{T}')$. By Lemma 2, the treewidth of $\mathcal{G}_\Pi$ is in $\mathcal{O}(k')$, which is in $\mathcal{O}(k)$. Assume towards a contradiction that consistency of Π can be decided in time $2^{o(k \cdot \log(k))} \cdot \text{poly}(|\text{at}(\Pi)|)$. By correctness of R (Theorem 3), this solves (G, P), contradicting Proposition 1. □

Our reduction works by construction for any pair-connected TD. Consequently, this immediately yields a lower bound for *pathwidth*, which is similar to treewidth, but admits only *path decompositions* (TDs whose tree is just a path).

Corollary 3 (Pathwidth lower bound)**.** *Given any normal or HCF program Π, where k is the pathwidth of the primal graph of Π. Then, unless ETH fails, the consistency problem for Π cannot be solved in time $2^{o(k \cdot \log(k))} \cdot \text{poly}(|\text{at}(\Pi)|)$.*

From Theorem 4, we follow that a general reduction from normal or HCF programs to SAT formulas can probably not avoid the treewidth (pathwidth) overhead, which renders our reduction from the previous section ETH-tight.

Corollary 4 (ETH-tightness of the Reduction to SAT)**.** *Under ETH, the increase of treewidth of the reduction using Formulas (1) to (6) cannot be significantly improved.*

Proof. Assume towards a contradiction that one can reduce from an arbitrary normal ASP program Π, where k is the treewidth of $\mathcal{G}_\Pi$ to a SAT formula, whose treewidth is in $o(k \cdot \log(k))$. Then, this contradicts Theorem 4, as we can use an algorithm (Samer and Szeider 2010; Fichte, Hecher, and Zisser 2019) for SAT being single exponential in the treewidth, thereby deciding consistency of Π in time $2^{o(k \cdot \log(k))} \cdot \text{poly}(|\text{at}(\Pi)|)$. □

5 Discussion, Conclusion, and Future Work

The curiosity of studying and determining the hardness of ASP and the underlying reasons has attracted the attention of the KR community for a long time. This paper discusses this question from a different angle, which hopefully will provide new insights into the hardness of ASP and foster follow-up work. The results in this paper indicate that, at least from a structural point of view, deciding the consistency of ASP is already harder than SAT, since ASP programs might compactly represent structural dependencies within the formalism. More concretely, compiling the hidden structural dependencies of a program to a SAT formula, measured in terms of the well-studied parameter treewidth, most certainly causes a blow-up of the treewidth of the resulting formula. In the light of a known result (Atserias, Fichte, and Thurley 2011) on the correspondence of treewidth and the resolution width applied in SAT solving, this reveals that ASP consistency might be indeed harder than solving SAT. We further presented a reduction from ASP to SAT that is aware of the treewidth in the sense that the reduction causes not more than this inevitable blow-up of the treewidth in the worst-case.

The work in this paper gives rise to plenty of future work. On the one hand, we are currently working on the comparison of different treewidth-aware reductions to SAT and variants thereof, and how these variants perform in practice. Further, we are curious about treewidth-aware reductions to SAT, which preserve answer sets bijectively or are modular (Janhunen 2006). We hope this work might reopen the quest to study the correspondence of treewidth and ASP solving similarly to (Atserias, Fichte, and Thurley 2011) for SAT. Also investigating further structural parameters "between" treewidth and directed variants of treewidth could lead to new insights, since for ASP directed measures (Bliem, Ordyniak, and Woltran 2016) often do not yield efficient algorithms.

Acknowledgements

This work has been supported by the Austrian Science Fund (FWF), Grants P32830 and Y698, as well as the Vienna Science and Technology Fund, Grant WWTF ICT19-065. We would like to thank the reviewers for their detailed and valuable feedback as well as Stefan Woltran for his support. Special appreciation goes to Andreas Pfandler for early discussions as well as to Jorge Fandinno for spotting a technical issue in an earlier version of the paper.

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., and Dodaro, C. 2016. Completion of disjunctive logic programs. In *IJCAI*, 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.

Bannach, M., and Berndt, S. 2019. Practical access to dynamic programming on tree decompositions. *Algorithms* 12(8):172.

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. 2018. Single-shot epistemic logic program solving. In *IJCAI'18*, 1714–1720. ijcai.org.

Bidoít, N., and Froidevaux, C. 1991. Negation by default and unstratifiable logic programs. *Theoretical Computer 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 Artificial Intelligence and Applications*. IOS Press.

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'2016*, volume 285 of *FAIA*, 1105–1113. IOS Press.

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.

Bomanson, J., and Janhunen, T. 2013. Normalizing cardinality rules using merging and sorting constructions. In *LPNMR'13*, volume 8148 of *LNCS*, 187–199. Springer.

Bomanson, J.; Gebser, M.; Janhunen, T.; Kaufmann, B.; and Schaub, T. 2016. Answer set programming modulo acyclicity. *Fundam. Inform.* 147(1):63–91.

Bomanson, J. 2017. lp2normal - A normalization tool for extended logic programs. In *LPNMR'17*, volume 10377 of *LNCS*, 222–228. Springer.

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

Charwat, G., and Woltran, S. 2019. Expansion-based QBF solving on tree decompositions. *Fundam. Inform.* 167(1-2):59–92.

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.

Dell, H.; Komusiewicz, C.; Talmon, N.; and Weller, M. 2017. The pace 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *IPEC'17*, LIPIcs, 30:1—30:13. Dagstuhl Publishing.

Diestel, R. 2012. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer.

Downey, R. G., and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

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.

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.

Fichte, J. K.; Hecher, M.; Morak, M.; and Woltran, S. 2018. Exploiting treewidth for projected model counting and its limits. In *SAT'18*, volume 10929 of *LNCS*, 165–184. Springer.

Fichte, J. K.; Hecher, M.; and Zisser, M. 2019. An improved gpu-based SAT model counter. In *CP'19*, volume 11802 of *LNCS*, 491–509. Springer.

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.

Flum, J., and Grohe, M. 2006. *Parameterized Complexity Theory*, volume XIV of *Theoretical Computer Science*. Springer.

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

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014. Answer set programming as SAT modulo acyclicity. In *ECAI*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 351–356. IOS Press.

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.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Comput.* 9(3/4):365–386.

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.

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., and Niemelä, I. 2016. The answer set programming paradigm. *AI Magazine* 37(3):13–24.

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.

Kleine Büning, H., and Lettman, T. 1999. *Propositional logic: deduction and algorithms*. Cambridge University Press, Cambridge.

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

Lackner, M., and Pfandler, A. 2012. Fixed-parameter algorithms for finding minimal models. In *KR'12*. AAAI Press.

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

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.

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

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.

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.

Pichler, R.; Rümmele, S.; Szeider, S.; and Woltran, S. 2014. Tractable answer-set programming with weight constraints: bounded treewidth is not enough. *TPLP* 14(2).

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.

Schaub, T., and Woltran, S. 2018. Special issue on answer set programming. *KI* 32(2-3):101–103.

Scheffler, P. 1994. A Practical Linear Time Algorithm for Disjoint Paths in Graphs with Bounded Tree-width. Technical Report Report-396-1994, TU Berlin. Available at: http://www.redaktion.tu-berlin.de/fileadmin/i26/download/AG_DiskAlg/FG_KombOptGraphAlg/preprints/1994/Report-396-1994.ps.gz.

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