On the Structural Hardness of Answer Set Programming: Can Structure Efficiently Confine the Power of Disjunctions?*

Markus Hecher[1], **Rafael Kiesel**[2]

[1] Massachusetts Institute of Technology, Cambridge, USA
[2] TU Wien, Vienna, Austria
hecher@mit.edu, rkiesel@tuwien.ac.at

Abstract

Answer Set Programming (ASP) is a generic problem modeling and solving framework with a strong focus on knowledge representation and a rapid growth of industrial applications. So far, the study of complexity resulted in characterizing hardness and determining their sources, fine-grained insights in the form of dichotomy-style results, as well as detailed parameterized complexity landscapes. Unfortunately, for the well-known parameter treewidth disjunctive programs require double-exponential runtime under reasonable complexity assumptions. This quickly becomes out of reach. We deal with the classification of structural parameters for disjunctive ASP on the program's rule structure (incidence graph).

First, we provide a polynomial kernel to obtain single-exponential runtime in terms of vertex cover size, despite subset-minimization being not represented in the program's structure. Then we turn our attention to strictly better structural parameters between vertex cover size and treewidth. Here, we provide double-exponential lower bounds for the most prominent parameters in that range: treedepth, feedback vertex size, and cliquewidth. Based on this, we argue that unfortunately our options beyond vertex cover size are limited. Our results provide an in-depth hardness study, relying on a novel reduction from normal to disjunctive programs, trading the increase of complexity for an exponential parameter compression.

Introduction

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011; Gebser et al. 2012) is a prominent declarative modeling and solving framework, enabling to efficiently solve problems in knowledge representation and artificial intelligence. Indeed, in the last couple of years, ASP has been extended several times and it grew into a rich modeling language, where solvers like clasp (Gebser, Kaufmann, and Schaub 2009) or wasp (Alviano et al. 2019) are readily available. This makes ASP a suitable target language for many problems, e.g., (Balduccini, Gelfond, and Nogueira 2006; Niemelä, Simons, and Soininen 1999; Nogueira et al. 2001; Guziolowski et al. 2013; Schaub and Woltran 2018; Abels et al. 2019), where problems are encoded in a logic program, which is a set of rules whose solutions are called answer sets.

Unfortunately, in terms of computational complexity, there is a catch to ASP's modeling comfort. Indeed, while there exist NP-complete fragments (Bidoít and Froidevaux 1991; Marek and Truszczyński 1991; Ben-Eliyahu and Dechter 1994), the *consistency problem* of deciding whether a disjunctive program admits an answer set is inherently hard, yielding Σ_2^P-completeness (Eiter and Gottlob 1995), even if we restrict rule body sizes to a constant (Truszczynski 2011).

How do we deal with this high complexity? Parameterized complexity (Cygan et al. 2015; Niedermeier 2006; Downey and Fellows 2013; Flum and Grohe 2006), offers a framework, enabling to analyze a problem's hardness in terms of certain *parameter(s)*, which has been extensively applied to ASP (Gottlob, Scarcello, and Sideri 2002; Gottlob, Pichler, and Wei 2010; Lackner and Pfandler 2012; Fichte, Kronegger, and Woltran 2019). For ASP there is growing research on the prominent structural parameter *treewidth* (Jakl, Pichler, and Woltran 2009; Calimeri et al. 2016; Bichler, Morak, and Woltran 2020; Bliem et al. 2020; Eiter, Hecher, and Kiesel 2021). Intuitively, the measure treewidth enables the solving of numerous combinatorially hard problems in parts. This parameter indicates the maximum number of variables of these parts one has to investigate during problem solving. Unfortunately, there is still a catch: Under the *Exponential Time Hypothesis (ETH)* (Impagliazzo, Paturi, and Zane 2001), the evaluation of disjunctive programs is prohibitively expensive, being inherently *double exponential* in its treewidth (Hecher 2022). Indeed this is a bad worst case; 2^{2^k} is larger than the (est.) number of atoms in the universe, already for $k{=}9$.

This motivates our search for preferably small structural parameters that enable single-exponential runtimes, i.e., we aim for structural properties strong enough to be significantly more exploitable than treewidth. We thereby consider the program's rule structure (incidence graph representation) and we only study parameters smaller than the vertex cover size, which is the number of vertices needed to cover every edge of a graph. We focus on the following structural parameters that are *more generally* applicable than treewidth, but smaller than vertex cover size, and state their intuitive meaning[1].

- Treedepth: How close is the structure to being a star?
- Feedback vertex number: How many atoms or rules need to be removed to obtain an acyclic graph structure?

*Extended version at http://arxiv.org/abs/2402.03539

[1] For formal definitions, see the preliminaries.

- Pathwidth: How close is the structure to being a path?

Our results will also have consequences for the parameters

- Bandwidth: How "far" do structural dependencies reach, assuming atoms and rules are linearly ordered?
- Cutwidth: How well can we linearly order atoms and rules, aiming for minimizing structural dependencies between predecessors of an atom/rule and their successors?
- Cliquewidth: How close is the structure to being a clique?

This paper thereby asks the following:

- Can we evaluate disjunctive ASP in single-exponential time for structural measures more general than treewidth?
- What makes a disjunctive program's structure harder to exploit than the structure of normal programs?
- Can we leverage the hardness of disjunction by translating from normal programs, thereby trading for an exponential decrease of structural dependencies (parameters)?

Contributions. We address these questions via algorithms and novel reductions, thereby establishing the following.

- First, we show a single-exponential upper bound when considering vertex cover size as a parameter. This is challenging, since our algorithm works despite the implicit subset-minimization required by disjunctive ASP, which is, however not directly manifested in the structural representation of the program. We present a polynomial-sized kernel for disjunctive ASP, which then yields a single-exponential algorithm for computing answer sets.
- Then, we show how to reduce from normal ASP to (full) disjunctive ASP, thereby exponentially *decreasing the program's cyclicity* (feedback vertex size) from k to $\log(k)$. To the best of our knowledge, this is the first reduction that exponentially reduces structural dependency of logic programs, at the cost of solving a harder program fragment. This reduction leads to ETH-tight double-exponential lower bounds for feedback vertex size.[2]
- The idea of this reduction technique has many further consequences. Indeed, by a generalization of this concept we obtain tight double-exponential bounds for the structural parameter treedepth. Even further, with this idea, we rule out single-exponential algorithms for the structural measures pathwidth, bandwidth, cliquewidth, and cutwidth. Unfortunately, given these lower bounds for most prominent measures between vertex cover and treewidth as well as the observations of our algorithm for vertex cover we do not expect significantly better parameters yielding better runtimes – disjunction limits structural exploitability.

Related Work. For disjunctive ASP, algorithms have been proposed (Jakl, Pichler, and Woltran 2009; Pichler et al. 2014) running in time linear in the instance size, but double exponential in the treewidth. Hardness of problems has been studied by means of runtime dependency in the treewidth, e.g., levels of exponentiality, where triple-exponential algorithms are known (Lokshtanov, Marx, and Saurabh 2011; Marx and Mitsou 2016; Fichte, Hecher, and Pfandler 2020). For quantified Boolean formulas (QBFs) parameterized by vertex cover size, a single-exponential runtime result is known (Lampis and Mitsou 2017). However, this result does not trivially transfer to ASP, as a direct encoding of subset-minimization causes an unbounded increase of vertex cover size. Also, lower bounds for QBFs and some of our considered parameters are known (Pan and Vardi 2006; Lampis and Mitsou 2017; Fichte, Hecher, and Pfandler 2020; Fichte et al. 2023). We strengthen these results for ASP, providing ETH-tight bounds for many measures. Programs of bounded even or odd cycles have been analyzed (Lin and Zhao 2004). Further, the feedback width has been studied, which depends on the atoms required to break large SCCs (Gottlob, Scarcello, and Sideri 2002). For SAT, empirical results (Atserias, Fichte, and Thurley 2011) involving resolution-width and treewidth yield efficient SAT solver runs on instances of small treewidth.

[2]The results even hold for the smallest number of atoms that, upon removal from the program, yields a structure of (almost) paths.

Preliminaries

We assume familiarity with propositional satisfiability (SAT) (Biere et al. 2009; Kleine Büning and Lettman 1999).

Answer Set Programming (ASP). We follow standard definitions of propositional ASP (Brewka, Eiter, and Truszczyński 2011; Janhunen and Niemelä 2016). Let ℓ, m, n be non-negative integers such that $\ell \leq m \leq n$, $a_1, \ldots, a_n$ 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 $\mathrm{at}(\Pi) := \bigcup_{r\in\Pi} \mathrm{at}(r)$. Further, $\mathrm{pos}(r, x)$ refers to the unique *position* of $x \in \mathrm{at}(r)$ in r. The rules, whose head contain $a \in \mathrm{at}(\Pi)$ are given by $\mathcal{H}(a) := \{r \in \Pi \mid a \in H_r\}$. We define the completion $\mathcal{C}(\Pi)$ by $\Pi \cup \{r_1^h \vee \ldots \vee r_\ell^h \leftarrow h \mid h \in \mathrm{at}(\Pi), \mathcal{H}(h) = \{r_1, \ldots, r_\ell\}\} \cup \{\leftarrow r^h, \neg a, \leftarrow r^h, b, \mid r \in \Pi, a \in B_r^+, b \in B_r^- \cup H_r, b \neq h\}$ (Clark 1977). Program Π is *normal* if $|H_r| \leq 1$ for every $r \in \Pi$; Π is *normalized* if for every $r \in \Pi$, $|H_r \cup B_r^+ \cup B_r^-| \leq 3$. Normalization preserves hardness for known fragments (Truszczynski 2011). The *dependency graph* D_Π of Π is the directed graph on the atoms $\bigcup_{r\in\Pi} H_r \cup B_r^+$, 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). Program Π is *tight* if D_Π is acyclic.

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 $I \models \Pi$. The *Gelfond-Lifschitz (GL) reduct* of Π under I is the program Π^I obtained from Π by first removing all rules r with $B_r^- \cap I \neq \emptyset$ and then removing all $\neg z$ where $z \in B_r^-$ from the remaining rules r (Gelfond and Lifschitz 1991). I is an *answer set* of a program Π, denoted $I \models \Pi$, if I is a minimal model of Π^I. A tight program Π is *fully tight* if for every model M of Π, there is a unique model M' of $\mathcal{C}(\Pi)$ and vice versa, such that $M = M' \cap \mathrm{at}(\Pi)$. Intuitively, this means that for computing answer sets it is sufficient to compute the models of Π. Deciding whether an ASP program has an answer set is called

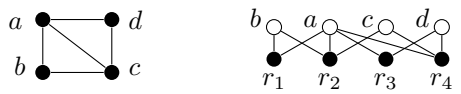

Figure 1: (Left): The primal graph $\mathcal{G}_{\Pi_1}$ of program Π_1 of Example 1. (Right): The incidence graph $\mathcal{I}_{\Pi_1}$ of Π_1.

consistency, which is Σ_2^{P}-complete (Eiter and Gottlob 1995). If the input is restricted to normal programs, the complexity drops to NP-complete (Marek and Truszczyński 1991).

Graph Representations. We assume familiarity with standard notions in computational complexity (Papadimitriou 1994) and graph theory, cf., (Diestel 2012). For parameterized complexity, we refer to standard texts (Cygan et al. 2015).

We need graph representations to deploy structural measures (Jakl, Pichler, and Woltran 2009). The *primal graph* $\mathcal{G}_\Pi$ of a program Π has the atoms of Π as vertices and an edge $\{a, b\}$ if there exists a rule $r \in \Pi$ with $a, b \in \mathrm{at}(r)$. The *incidence graph* $\mathcal{I}_\Pi$ of Π has as vertices the atoms and rules of Π and an edge $\{a, r\}$ for every rule $r \in \Pi$ with $a \in \mathrm{at}(r)$.

Example 1. *Consider the tight program* $\Pi_1{=}\{r_1, r_2, r_3, r_4\}$, *where* $r_1 = b \leftarrow \neg a$; $r_2 = b \leftarrow a, c$; $r_3 = a \vee d \leftarrow$; *and* $r_4 = c \leftarrow a, \neg d$. *Then,* Π_1 *admits two answer sets* $\{b, d\}, \{a, b, c\}$. *Figure 1 shows graph representations of* Π_1. *Program* $\Pi_2 = \Pi_1 \cup \{r_1^b \vee r_2^b \leftarrow b; \leftarrow r_1^b, a; \leftarrow r_2^b, \neg a; \leftarrow r_2^b, \neg c; \leftarrow a, d; \leftarrow c, \neg a; \leftarrow c, d\}$ *is* fully tight.

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

Bandwidth & Cutwidth. Let $G = (V, E)$ be a given graph and $f : V \to \{1, \ldots, |V|\}$ be a bijective (one-to-one) *ordering* that uniquely assigns a vertex to an integer. The *bandwidth* of G is the minimum $\max_{\{u,v\} \in E} |f(u) - f(v)|$ among every ordering f for G. The *cutwidth* of G is the minimum $\max_{1 \leq i \leq |V|} |\{\{x, y\} \in E \mid f(x) \leq i, f(y) > i\}|$ among every ordering f for G.

Cliquewidth. The *cliquewidth* of a graph G is the minimum number of labels needed to construct G via labeled graphs over the following 4 operations: (1) create a new vertex with label ℓ, (2) disjoint union of two labeled graphs, (3) create new edges between all vertices with label ℓ and those with ℓ', and (4) rename label ℓ to ℓ'.

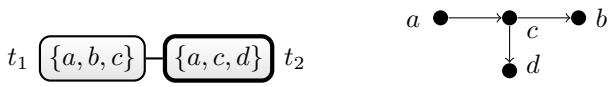

Figure 2: (Left): A TD (PD) $\mathcal{T}$ of $\mathcal{G}_{\Pi_1}$ of Figure 1. (Right): A Trémaux tree of $\mathcal{G}_{\Pi_1}$.

Feedback Vertex Size. A set S is a *feedback vertex set (FVS)* of a graph G, if the graph obtained from G after removing S is acyclic. S is *sparse* (for a program Π) if (i) $G \in \{\mathcal{G}_\Pi, \mathcal{I}_\Pi\}$, and (ii) for every two atoms $a, b \in \mathrm{at}(\Pi) \setminus S$, Π contains at most one rule over both a, b. The *feedback vertex size* of G is the smallest $|S|$ among every FVS S of G.

Treedepth. A *Trémaux tree* T of a graph $G = (V, E)$ is a rooted tree such that for every edge $\{u, v\} \in E$, either u is an ancestor of v in T or vice versa. The *treedepth* of G is the minimum *height* among all Trémaux trees of G.

Vertex Cover Size. A *vertex cover* of a graph $G = (V, E)$ is a set $S \subseteq V$ such that for every $\{u, v\} \in E$, we have $u \in S$ or $v \in S$. The *vertex cover size* of G is the smallest $|S|$ over all vertex covers S of G.

Example 2. *Recall* Π_1 *from Example 1. Figure 2 (left) depicts a PD of* $\mathcal{G}_{\Pi_1}$, *whose width is* 2, *corresponding to the pathwidth, treewidth, and bandwidth of the graph. The cutwidth of* $\mathcal{G}_{\Pi_1}$ *is* 3 *and its cliquewidth is* 2, *as two labels are sufficient to construct* $\mathcal{G}_{\Pi_1}$. *The smallest FVSs of* $\mathcal{G}_{\Pi_1}$ *are* $\{a\}$ *and* $\{c\}$; *the smallest sparse FVS is* $S_1 = \{a, c\}$. *Further,* S_1 *is the smallest vertex cover of* $\mathcal{G}_{\Pi_1}$. *Figure 2 (right) shows a Trémaux tree of* $\mathcal{G}_{\Pi_1}$ *of height* 2 *(treedepth of* $\mathcal{G}_{\Pi_1}$*).*

Vertex Cover Limits the Cost of Disjunction

In (Lampis and Mitsou 2017), the authors showed that satisfiability of quantified Boolean formulas (QBFs) ϕ is possible in single exponential time in the size of the smallest vertex cover of the primal graph of ϕ, if the size of clauses is bounded by a constant independent of ϕ.

This begs the question of whether the same is possible for ASP. The first idea that comes to mind, is to translate programs Π to QBFs $\phi(\Pi)$. However, using the standard translation (Egly et al. 2000), the smallest vertex cover of the program $\phi(\Pi)$ has at least size $|\mathrm{at}(\Pi)|$, and thus, cannot be bounded in the size of the smallest vertex cover for Π.

We show something stronger instead, namely, for programs with bounded rule size c, e.g., $c = 3$ for normalized programs, we give a polynomial kernel in terms of vertex cover size.

Theorem 1 (Polynomial VC-Kernel)**.** *Let* Π *be a program such that for every rule* $r \in \Pi$ *we have* $|H_r| + |B_r^-| + |B_r^+| \leq c$ *for* $c \in \mathbb{N}$. *Further, let* $S \subseteq \mathrm{at}(\Pi)$ *be a vertex cover of* $\mathcal{G}_\Pi$. *Then there is a program* Π' *where we have (i)* $|\mathrm{at}(\Pi')| \leq 4^c \binom{|S|}{c}$ *and (ii)* Π *is consistent iff* Π' *is consistent.*

Proof (Sketch). The *role* of an atom $a \notin S$ in the program is defined by the rules where it occurs in. Since S is a vertex cover of $\mathcal{G}_\Pi$, all other atoms in the rules that contain a must be in S. Because the rule size is bounded by c, we can bound the number of different roles. However, we can show that if two atoms have the same role, then we can remove one of them without changing consistency of the program. □

A similar result holds if the primal graph does not directly represent the structure of *type 1* and *type 2* rules of the form $a \leftarrow \neg b$ or $a \vee b \leftarrow$, respectively. This is important, as these rules are oftentimes required to model "guesses" or use the full power of ASP semantics, respectively. Indeed, this will otherwise be the source of a large vertex cover size. To this end, let the $\mathcal{G}^i_\Pi$, for $i = 0, 1, 2$ be the primal graphs consisting of the type i rules of a program Π, where type 0 rules are rules that are not of type 1 or 2.

Corollary 1. *Let Π be a program such that for every rule $r \in \Pi$ we have $|H_r| + |B^-_r| + |B^+_r| \leq c$ for $c \in \mathbb{N}$, where every atom has at most one neighbor in $\mathcal{G}^1_\Pi$ or $\mathcal{G}^2_\Pi$ (that is not a neighbor in $\mathcal{G}^0_\Pi$). Furthermore, let $S \subseteq \mathrm{at}(\Pi)$ be a vertex cover of $\mathcal{I}^0_\Pi$. Then there exists a program Π' such that (i) $|\mathrm{at}(\Pi')| \leq 3 \cdot 4 \cdot (4^c \binom{|S|}{c})^2 + 4^c \binom{|S|}{c}$ and (ii) Π is consistent iff Π' is consistent.*

The algorithm above also works for the incidence graph.

Corollary 2. *Let Π be a program such that for every rule $r \in \Pi$ we have $|H_r| + |B^-_r| + |B^+_r| \leq c$ for $c \in \mathbb{N}$ and let $S \subseteq \mathrm{at}(\Pi)$ be a vertex cover of $\mathcal{I}_\Pi$. Then there exists a program Π' such that (i) $|\mathrm{at}(\Pi')| \leq 4^c \binom{|S|}{c} c^c$ and (ii) Π is consistent iff Π' is consistent.*

The polynomial kernels above immediately give rise to the following runtime result.

Corollary 3. *Let Π be a program such that for every rule $r \in \Pi$ we have $|H_r| + |B^-_r| + |B^+_r| \leq c$ for $c \in \mathbb{N}$ and vertex cover size k of the primal graph. Then we can decide consistency of Π in time $\mathcal{O}(2^{2 \cdot 4^c \binom{k}{c}} \mathrm{poly}(|\mathrm{at}(\Pi)|))$.*

Decrease Cyclicity and Treedepth by the Power of Disjunction

In this section we show how to translate a (tight) logic program to a disjunctive program, thereby exponentially decreasing the program's cyclicity, i.e., it's feedback vertex size and explicitly utilizing the structural power of disjunctive ASP. The reduction uses head-cycles and disjunction in order to carry out the exponential parameter decrease, which will yield new lower bounds for disjunctive ASP. In fact, we even prove a stronger lower bound, which already holds for (i) normalized fully tight programs and (ii) sparse FVS size.

The idea of this reduction $\mathcal{R}_{fvs}$ is as follows. We take a normalized fully tight program Π and a sparse feedback vertex set S of $\mathcal{G}_\Pi$ of size $|S| = k$. From this we construct a disjunctive program, where we decouple the atoms of Π and we model three pointers to S to check satisfiability of Π. Since Π is normalized, i.e., ever rule contains at most three atoms, indeed three pointers are sufficient to refer to all the atoms of a rule. These three pointers can be expressed by using $3 \log(k)$ many atoms, which will allow us to achieve the desired compression of feedback vertex size from k to $\log(k)$.

This reduction $\mathcal{R}_{fvs}$ constructs a disjunctive program, thereby relying on the atoms $\mathrm{at}(\Pi)$ of the given (normalized) fully tight program Π and the following additional auxiliary atoms. We use atoms sat to indicate satisfaction for the interpretation over $\mathrm{at}(\Pi)$ of the program. Further, for every $r \in \Pi$, if sat_r holds, satisfaction of r shall be checked. Further, we require bit atoms of the form b^i_j for three pointers ($1 \leq j \leq 3$), which indicates that the i-th bit of pointer j is set to one, and we need $\lceil \log(|S|) \rceil$ many of these bits, i.e., $0 \leq i < \lceil \log(|S|) \rceil$. In order to also unset these atoms (i.e., set the bits to zero), we require corresponding inverse copies of the form $\overline{b^i_j}$, with the intended meaning that the i-th bit of pointer j is set to zero. Intuitively, a subset over these atoms of cardinality $\lceil \log(|S|) \rceil$ for a fixed j addresses exactly one atom that is contained in S. We briefly need to define auxiliary notation. For every atom $x \in S$, we let $[\![x]\!]_j$ be the set of these bit atoms for pointer j of cardinality $\lceil \log(|S|) \rceil$ that uniquely addresses x. To this end, one may assume that the elements in S are given in any arbitrary, but fixed order and that $[\![x]\!]_j$ then binary-encodes the ordinal position of x in S. Finally, in addition to the capability of referring to atoms in S, we also need to set the value for the target of the three pointers, resulting in atoms v_1, v_2, and v_3.

The Reduction. We are ready to proceed with the formal description of our reduction, as given in Figure 3[3]. To this end, we take our normalized fully tight program Π and the feedback vertex set S of $\mathcal{G}_\Pi$ and construct a program Π' according to $\mathcal{R}_{fvs}$ as follows. For improved readability, reduction $\mathcal{R}_{fvs}$ is split into four different groups. In the first group, consisting of Rules (1),we guess the interpretation, the bits that will form pointers, the pointer target's values, as well as the rules of Π that shall be checked. The atoms appearing under disjunction will be subject to saturation, so any answer set can only contain all of these atoms, which is ensured by Rules (2)–(3) of the second group, where (3) requires sat to be in every answer set. So, whenever we check an answer set candidate M, for *every combination* of these atoms under disjunction, there must not exist a $\subseteq$-smaller model of Π^M, i.e., intuitively, these atoms are implicitly universally quantified.

The remainder boils down to synchronizing pointers and their values, which is achieved in the third group of rules, and checking rules in the fourth group. Rules (4) ensure that whenever we need to check rule r, i.e., in case sat_r holds, as soon as any bit of the three pointers is not in accordance with the bit combination addressing the three atoms of r, we obtain sat. In other words, as soon as we refer to the atoms of r incorrectly, we just skip those interpretations. Similarly, by Rules (5), we have that whenever the j-pointer refers to an atom x that is contained in S and the pointer value v_j is contained in the answer set candidate, but x is not (or vice versa), we also obtain sat. Note that it is crucial for the feedback vertex set of the resulting program that sat_r does not appear in any of the two rules.

Finally, a rule r is satisfied if the pointer value v_j for the j-th atom is set accordingly, ensured by Rules (6) and (7), or if any of those rule atoms not in S are set accordingly, see Rules (8) and (9). The only remaining rule is (10), which is required to avoid $\subseteq$-smaller reduct models containing every atom of the form $\overline{sat_r}$ and none of the form sat_r. Intuitively, this is essential, since not checking any of these rules r would lead to a $\subseteq$-smaller reduct model.

Before we discuss the consequences of this rule in more details, and we briefly demonstrate the reduction below.

[3]For brevity, $\dot{b}$ for atom b refers to $\overline{b^i_j}$ if b=b^i_j and to b^i_j if b=$\overline{b^i_j}$.

Guess Interpretation, Pointers, and Values

$$x \vee \overline{x} \leftarrow \qquad sat_r \vee \overline{sat_r} \leftarrow \qquad b_j^i \vee \overline{b_j^i} \leftarrow \qquad v_j \vee \overline{v_j} \leftarrow \quad \text{for every } x \in \text{at}(\Pi), r \in \Pi, 0 \leq i < \lceil \log(|S|) \rceil, 1 \leq j \leq 3 \quad (1)$$

Saturate Pointers and Values

$$b_j^i \leftarrow sat \qquad \overline{b_j^i} \leftarrow sat \qquad v_j \leftarrow sat \qquad \overline{v_j} \leftarrow sat \qquad \text{for every } 0 \leq i < \lceil \log(|S|) \rceil, 1 \leq j \leq 3 \quad (2)$$

$$sat_r \leftarrow sat \qquad \overline{sat_r} \leftarrow sat \qquad \leftarrow \neg sat \qquad \text{for every } r \in \Pi \quad (3)$$

Synchronize Pointers and Values

$$sat \leftarrow sat_r, \dot{b} \qquad \text{for every } r \in \Pi, 1 \leq j \leq 3, x \in S \cap \text{at}(r), j = \text{pos}(r, x), b \in [\![x]\!]_j \quad (4)$$

$$sat \leftarrow b^0, \ldots, b^n, \overline{v_j}, x \qquad sat \leftarrow b^0, \ldots, b^n, v_j, \overline{x} \qquad \text{for every } 1 \leq j \leq 3, x \in S, [\![x]\!]_j = \{b^0, \ldots, b^n\} \quad (5)$$

Check Satisfiability of Rules

$$sat \leftarrow sat_r, v_j \qquad \text{for every } r \in \Pi, x \in S \cap (H_r \cup B_r^-), j = \text{pos}(r, x) \quad (6)$$

$$sat \leftarrow sat_r, \overline{v_j} \qquad \text{for every } r \in \Pi, x \in S \cap B_r^+, j = \text{pos}(r, x) \quad (7)$$

$$sat \leftarrow sat_r, x \qquad \text{for every } r \in \Pi, x \in (H_r \cup B_r^-) \setminus S \quad (8)$$

$$sat \leftarrow sat_r, \overline{x} \qquad \text{for every } r \in \Pi, x \in B_r^+ \setminus S \quad (9)$$

$$sat \leftarrow \overline{sat_{r_1}}, \ldots \overline{sat_{r_n}} \qquad \text{where } \Pi = \{r_1, \ldots, r_n\} \quad (10)$$

Figure 3: The reduction $\mathcal{R}_{fvs}$ that takes a normalized fully tight program Π and a sparse feedback vertex set S of $\mathcal{G}_\Pi$.

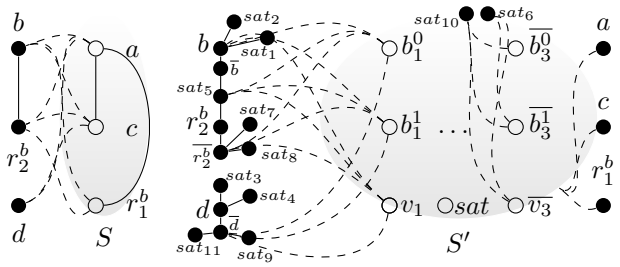

Figure 4: Visualization of the structure of $\mathcal{R}_{fvs}$ for some normalized fully tight program Π. (Left): Primal graph $\mathcal{G}_\Pi$ together with a sparse FVS S that connects $\mathcal{G}_\Pi$. (Right): $\mathcal{G}_{\Pi'}$ (simplified) and a sparse FVS S' of Π', obtained by $\mathcal{R}_{fvs}$.

Example 3. *Recall program Π_2 over 6 atoms and 11 rules, and S_1 from Example 1. Observe that, e.g., $S_2 = S_1 \cup \{r_1^b\}$ is a sparse FVS of $\mathcal{G}_{\Pi_2}$. Below we list selected rules of $\mathcal{R}_{fvs}(\Pi_2, S_2)$. Since $|S_2| \leq 3$, we require $3 \cdot \lceil \log(|S_2|) \rceil = 6$ bit atoms b_j^0, b_j^1 and 3 values v_j ($1 \leq j \leq 3$), to simultaneously address and assign up to 3 atoms in a rule of Π_2.*

Assuming $\text{pos}(r_4, a) = 1$ and $\text{pos}(r_4, c) = 2$, as well as $[\![a]\!]_1 = \{\neg b_1^0, \neg b_1^1\}$ and $[\![c]\!]_2 = \{\neg b_2^0, b_2^1\}$, we construct these rules for r_4: Besides Rules (1)–(3) and (10), Rules (4) ensure that if sat_{r_4} holds, we address a and c by $sat \leftarrow sat_{r_4}, b_1^0$; $sat \leftarrow sat_{r_4}, b_1^1$; $sat \leftarrow sat_{r_4}, b_2^0$; and $sat \leftarrow sat_{r_4}, \neg b_2^1$. Pointer values are in sync with atoms in S by Rules (5). Rules (6)–(9) ensure satisfiability of r_4 by $sat \leftarrow sat_{r_4}, \neg v_1$; $sat \leftarrow sat_{r_4}, v_2$; and $sat \leftarrow sat_{r_4}, d$.

Figure 4 visualizes the relation between a primal graph $\mathcal{G}_\Pi$ and the resulting primal graph $\mathcal{G}_{\mathcal{R}_{fvs}(\Pi,S)}$, along the lines of our running example where $\Pi = \Pi_2$ and $S = S_2$.

Results. First, we show runtime results and correctness.

Lemma 1 (Runtime & Correctness)**.** *Let Π be any normalized fully tight program, S be any sparse FVS of $\mathcal{G}_\Pi$, and let $\Pi' = \mathcal{R}_{fvs}(\Pi, S)$. $\mathcal{R}_{fvs}$ runs in time $\mathcal{O}(|\text{at}(\Pi)| + |\Pi| \cdot \log(|S|) + |S|)$ and it is correct: Any answer set M of Π can be extended to an answer set M' of Π'. Further, for any answer set M' of Π', $M' \cap \text{at}(\Pi)$ is an answer set of Π.*

Leaving Rule (10) aside, we can show that the reduction significantly decreases the feedback vertex size. Interestingly, while the disjunction over atoms at(Π) in Rules (1) is not required for saturation, it serves in preserving sparsity.

Theorem 2. *Let Π be any normalized fully tight program, S be any sparse FVS of $\mathcal{G}_\Pi$, and let $\Pi' = \mathcal{R}_{fvs}(\Pi, S)$. Then, there is a sparse FVS of $\mathcal{G}_{\Pi''}$ of size $\mathcal{O}(\lceil \log(|S|) \rceil)$, where Π'' is a normalized program of $\Pi' \setminus \{r\}$, with r being Rule (10).*

We do not expect that this result can be improved without excluding Rule (10). Intuitively, the problem is that normalizing this rule significantly incrases the feedback vertex size. However, for a given sparse feedback vertex set S of the primal graph $\mathcal{G}_\Pi$, we have a direct relationship between $|S|$ and the feedback vertex size of the natural incidence graph $\mathcal{I}_{\Pi'}$ of the resulting program Π'. We obtain the following result.

Lemma 2 (Compression)**.** *Let Π be a normalized fully tight program, S be a sparse FVS of $\mathcal{G}_\Pi$, and $\Pi' = \mathcal{R}_{fvs}(\Pi, S)$. Then, there is a sparse FVS of $\mathcal{I}_{\Pi'}$ of size $\mathcal{O}(\lceil \log(|S|) \rceil)$.*

Proof. We define $S'' := S' \cup \{r\}$, where S' is defined in Lemma 2 and r refers to Rule (10), which will be a FVS of $\mathcal{I}_{\Pi'}$ for the program Π'. Obviously, S'' breaks-up the cycles in $\mathcal{I}_{\Pi'}$ caused by Rule (10). Further, since by Theorem 2, S' is a sparse FVS of $\mathcal{G}_{\Pi'}$, we follow that removing S'' from $\mathcal{I}_{\Pi'}$ results in an acyclic graph. □

Note that Rules (5) can be easily normalized without increasing the feedback vertex size. The result above allows us to directly prove the following lower bound.

Theorem 3 (FVS LB)**.** *Let* Π *be any disjunctive program. Under ETH the consistency of* Π *cannot be decided in time* $2^{2^{o(k)}} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$ *for (sparse) FVS size* k *of* $\mathcal{I}_\Pi$.

Proof. Assume towards a contradiction that despite ETH, we can decide Π in time $2^{2^{o(k)}} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$. Then, we take any 3-CNF formula F and transform it into a normalized fully tight program $\Pi' := \{a \leftarrow \neg\overline{a}, \overline{a} \leftarrow \neg a \mid a \in \mathrm{var}(F)\} \cup \{\leftarrow \hat{\ell}_1, \hat{\ell}_2, \hat{\ell}_3 \mid (\ell_1 \vee \ell_2 \vee \ell_3) \in F\}$, where $\hat{\ell} := \neg\ell$ if $\ell \in \mathrm{var}(F)$ and $\hat{\ell} := \ell$ otherwise. We compute a sparse FVS S of $\mathcal{G}_{\Pi'}$ in time $3.619^{\mathcal{O}(k)} \cdot \mathrm{poly}(|\mathrm{at}(\Pi')|)$ (Kociumaka and Pilipczuk 2014). We apply our reduction and construct $\Pi := \mathcal{R}_{fvs}(\Pi', S)$, running in polynomial time. By Lemma 2, there is a FVS of $\mathcal{I}_\Pi$ of size $k := \mathcal{O}(k)$ with $k := \log(|S|)$. Solving Π in time $2^{2^{o(k)}} \cdot \mathrm{poly}(|\mathrm{at}(\Pi')|)$ implies solving F in time $2^{o(|S|)} \cdot \mathrm{poly}(|\mathrm{var}(F)|)$, which contradicts ETH. □

We get a similar result for the primal graph, assuming that just a single rule is not represented in this graph.

The lower bound can be strengthened to the *distance* k *to almost paths*, even if the path lengths are linear in k. This distance *counts the number of vertices* that need to be removed from a graph, to obtain disconnected paths, where every vertex may have one additional degree-1 neighbor.

Theorem 4. *Let* Π *be any disjunctive program. Under ETH, the consistency of* Π *cannot be decided in time* $2^{2^{o(k)}} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$ *for distance* k *to almost paths of* $\mathcal{I}_\Pi$. *The result still holds if the largest path length is in* $\mathcal{O}(k)$.

Even further, we also get a tight lower bound for treedepth.

Theorem 5 (TDP LB)**.** *Let* Π *be any disjunctive program. Under ETH the consistency of* Π *cannot be decided in time* $2^{2^{o(k)}} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$ *for treedepth* k *of* $\mathcal{I}_\Pi$.

A Note on Normal Programs. Our reductions still work for non-tight programs, assuming that besides Clark's completion (Clark 1977), a suitable technique for treating positive cycles (Lin and Zhao 2003), e.g., a treewidth-aware (Hecher 2022) level ordering (Janhunen 2006), and normalization, has been applied. However, for separating disjunctive from normal programs regarding structural hardness, normalized fully tight programs are aready sufficient.

Hardness for Bandwidth and Cutwidth

The reduction of the previous section can be further generalized to work also for other parameters more general than treewidth. To show double-exponential lower bounds for these parameters, we design a reduction on tree decompositions. Then we will obtain stronger results by tweaking the reduction and considering more restricted decompositions.

For the ease of presentation, we rely on the following TDs.

Definition 1 (Annotated TD)**.** *Let* Π *be a program,* G_Π *be a graph representation of* Π*, and* $\mathcal{T} = (T, \chi)$ *be a nice TD of* G_Π *such that* $T = (V, E)$. *Further, let* $\varphi : (\mathrm{at}(\Pi) \cup \Pi) \to V$ *be an injective mapping such that (i) for every* $a \in \mathrm{at}(\Pi)$, *we have* $a \in \chi(\varphi(a))$ *and (ii) for every* $r \in \Pi$, *we have* $\mathrm{at}(r) \subseteq \chi(\varphi(r))$. *Then, we call* (T, χ, φ) *an* annotated TD.

Note that an annotated TD ensures that every atom and rule get assigned a unique node, whose bag contains the atom and the rule's atoms, respectively. Any TD can be annotated by greedily assigning suitable atoms and rules to a node and, if necessary, duplicating this node to ensure injectivity.

The Reduction. Conceptually, the ideas of our reduction $\mathcal{R}_{td}$ for exponentially decreasing treewidth are similar to above. However, we require different auxiliary atoms. To this end, consider a normalized fully tight program Π and an annotated nice TD $\mathcal{T} = (T, \chi, \varphi)$ of $\mathcal{G}_\Pi$. Then, we use bit atoms of the form $b^i_{t,j}$ and $\overline{b^i_{t,j}}$, i.e., we require three pointers for each node t of T, where $1 \leq j \leq 3$ and $0 \leq i < \lceil \log(|\chi(t)|) \rceil$. A consistent subset over these bit atoms for a fixed t and j addresses (targets at) exactly one atom contained in $\chi(t)$. Hence, to represent the value of this target, we require corresponding value atoms $v_{t,j}$.

In order to avoid unintentional increases of treewidth, we implicitly split Rule (10) and guide it along $\mathcal{T}$, from the leaves towards the root of T. In addition to atoms $sat_r, \overline{sat_r}$ for a rule $r \in \Pi$, atoms sat_t store the evaluation of parts of Rule (10) up to t. So, we do not need atom sat and instead use sat_{t^*} where $t^* = \mathrm{root}(T)$. Finally, for every atom $x \in \chi(t)$, we let $[\![x]\!]_{t,j}$ be the set of bit atoms for the j-th pointer that uniquely addresses x, assuming (as above) that $[\![x]\!]_{t,j}$ uniquely binary-encodes the ordinal position of x in $\chi(t)$.

Reduction $\mathcal{R}_{td}$ is given in Figure 5. Compared to the previous reduction, we synchronize atoms x with pointer values in Rules (15) and (16), where, assuming that $t = \varphi(x)$, we preserve the exponential decrease of treewidth by synchronizing only one atom in $\chi(t)$ (i.e., not more than $\log(k)$ bits) at once. Further, Rules (17) synchronize neighboring pointers (bit atoms) and bit value atoms. Satisfiability of a rule is verified per bag, using Rules (18), (19). Finally, Rules (20) guide Rule (10) along $\mathcal{T}$, thereby evaluating whether all atoms $\overline{sat_r}$ hold for rules $r \in \Pi$, where $\varphi(r)$ maps to a node below t.

Results. As above, we prove runtime and correctness next.

Lemma 3 (Runtime & Correctness)**.** *Let* Π *be any normalized fully tight program,* $\mathcal{T}$ *be an annotated nice TD of* $\mathcal{G}_\Pi$ *of width* k, *and let* $\Pi' = \mathcal{R}_{td}(\Pi, \mathcal{T})$. $\mathcal{R}_{td}$ *runs in time* $\mathcal{O}((|\mathrm{at}(\Pi)| + |\Pi|) \cdot k)$ *and is correct: Any answer set* M *of* Π *can be extended to an answer set* M' *of* Π'. *Also, for any answer set* M' *of* Π', $M' \cap \mathrm{at}(\Pi)$ *is an answer set of* Π.

Indeed, $\mathcal{R}_{td}$ exponentially decreases treewidth.

Lemma 4 (TW Compression)**.** *Let* Π *be any normalized fully tight program,* $\mathcal{T} = (T, \chi, \varphi)$ *be any annotated TD of* $\mathcal{G}_\Pi$ *of width* k, *and* $\Pi' = \mathcal{R}_{td}(\Pi, \mathcal{T})$. *There is a TD of* $\mathcal{G}_{\Pi''}$ *of width* $\mathcal{O}(\lceil \log(k) \rceil)$, *where* Π'' *is a normalized program of* Π'.

Since compression works for tree decompositions, it also works for path decompositions, yielding the following result.

Theorem 6 (PW/CLW LB)**.** *Let* Π *be any normalized disjunctive program. Under ETH the consistency of* Π *cannot be decided in time* $2^{2^{o(k)}} \cdot \mathrm{poly}(|\mathrm{at}(\Pi)|)$, *where* k *is the pathwidth or the cliquewidth of* $\mathcal{G}_\Pi$ *or* $\mathcal{I}_\Pi$.

Guess Interpretation, Pointers, and Values

$$x \vee \overline{x} \leftarrow \quad sat_r \vee \overline{sat_r} \leftarrow \quad b^i_{t,j} \vee \overline{b^i_{t,j}} \leftarrow \quad v_{t,j} \vee \overline{v_{t,j}} \leftarrow \quad \text{for every } t \text{ in } T, r{\in}\Pi, x{\in}\text{at}(\Pi), 0{\leq}i{<}\lceil\log(|\chi(t)|)\rceil, 1{\leq}j{\leq}3 \quad (11)$$

Saturate Pointers and Values

$$b^i_{t,j} \leftarrow sat_{\text{root}(T)} \quad \overline{b^i_{t,j}} \leftarrow sat_{\text{root}(T)} \quad \leftarrow \neg sat_{\text{root}(T)} \qquad \text{for every } t \text{ in } T, 0 \leq i < \lceil\log(|\chi(t)|)\rceil, 1 \leq j \leq 3 \quad (12)$$

$$v_{t,j} \leftarrow sat_{\text{root}(T)} \quad \overline{v_{t,j}} \leftarrow sat_{\text{root}(T)} \quad sat_r \leftarrow sat_{\text{root}(T)} \quad \overline{sat_r} \leftarrow sat_{\text{root}(T)} \qquad \text{for every } t \text{ in } T, r \in \Pi, 1 \leq j \leq 3 \quad (13)$$

Synchronize Pointers and Values

$$sat_{\text{root}(T)} \leftarrow sat_r, \dot{b} \qquad \text{for every } t \text{ in } T, t = \varphi(r), x \in \text{at}(r), j = \text{pos}(r, x), b \in [\![x]\!]_{t,j} \quad (14)$$

$$sat_{\text{root}(T)} \leftarrow b^0, \ldots, b^n, \overline{v_{t,j}}, x \qquad \text{for every } t \text{ in } T, t = \varphi(x), 1 \leq j \leq 3, [\![x]\!]_{t,j} = \{b^0, \ldots, b^n\} \quad (15)$$

$$sat_{\text{root}(T)} \leftarrow b^0, \ldots, b^n, v_{t,j}, \overline{x} \qquad \text{for every } t \text{ in } T, t = \varphi(x), 1 \leq j \leq 3, [\![x]\!]_{t,j} = \{b^0, \ldots, b^n\} \quad (16)$$

$$sat_{\text{root}(T)} \leftarrow b^0_t, \ldots, b^n_t, \dot{b_{t'}} \qquad \text{for every } t \text{ in } T, t' \in \text{chldr}(t), x \in \chi(t) \cap \chi(t'), 1 \leq j \leq 3,$$

$$sat_{\text{root}(T)} \leftarrow \overline{v_{t,j}}, v_{t',j} \qquad sat_{\text{root}(T)} \leftarrow v_{t,j}, \overline{v_{t',j}} \qquad [\![x]\!]_{t,j}{=}\{b^0_t, \ldots, b^n_t\}, b_{t'}{\in}[\![x]\!]_{t',j} \quad (17)$$

Check Satisfiability of Rules

$$sat_{\text{root}(T)} \leftarrow sat_r, v_{t,j} \qquad \text{for every } t \text{ in } T, t = \varphi(r), x \in (H_r \cup B^-_r), j = \text{pos}(r, x) \quad (18)$$

$$sat_{\text{root}(T)} \leftarrow sat_r, \overline{v_{t,j}} \qquad \text{for every } t \text{ in } T, t = \varphi(r), x \in B^+_r, j = \text{pos}(r, x) \quad (19)$$

$$sat_t \leftarrow sat_{t_1}, \ldots, sat_{t_o}, \overline{sat_{r_1}}, \ldots, \overline{sat_{r_m}} \qquad \text{for every } t \text{ in } T, \text{chldr}(t){=}\{t_1, \ldots, t_o\}, \{r{\in}\Pi \mid t{=}\varphi(r)\}{=}\{r_1, \ldots, r_m\} \quad (20)$$

Figure 5: The reduction $\mathcal{R}_{td}$ that takes a normalized fully tight program Π and an annotated nice TD $\mathcal{T} = (T, \chi, \varphi)$ of $\mathcal{G}_\Pi$.

Further, we can show stronger bounds than pathwidth by linking a more restricted variant of pathwidth to bandwidth.

Lemma 5. *Let $G = (V, E)$ be a graph, $\mathcal{T} = (T, \chi)$ be a PD of G of width k, s.t. every vertex in V occurs in at most two bags of $\mathcal{T}$. Then, the bandwidth of G is bounded by $2k - 1$.*

Theorem 7 (BW/CW LB)**.** *Let Π be any normalized disjunctive program. Under ETH, consistency of Π cannot be decided in $2^{2^{o(k)}} \cdot \text{poly}(|\text{at}(\Pi)|)$ for bandwidth k of $\mathcal{G}_\Pi$ ($\mathcal{I}_\Pi$).*

While bandwidth bounds the degree to obtain a tight bound for cutwidth, we require a *constant* degree (independent from cutwidth). However, we add auxiliary atom copies to ensure constant degree while *linearly* preserving the parameter.

Theorem 8 (CW LB)**.** *Let Π be any normalized disjunctive program. Under ETH the consistency of Π cannot be decided in time $2^{2^{o(k)}} \cdot \text{poly}(|\text{at}(\Pi)|)$ for cutwidth k of $\mathcal{G}_\Pi$ or $\mathcal{I}_\Pi$.*

Discussion and Conclusion

In this paper we discussed the computational effort of disjunctive ASP in terms of structural measures. Unfortunately, for *treewidth*, under reasonable assumptions (ETH), there is no hope for a runtime significantly better than *double-exponential* in treewidth. What about other parameters?

It turns out that for normalized programs and if we take a vertex cover of the program's rule structure (incidence graph), we obtain a kernel that is polynomial in the vertex cover's size. Note that this even holds despite the implicit subset-minimization, which, when translating into a logical formula causes the duplication of atoms, i.e., the formula admits only huge vertex covers. Hence, we achieve a *single-exponential* algorithm for disjunctive ASP. While this is good news, some practical programs admit only huge vertex covers in their structure. Is there a useful measure smaller than vertex cover that still admits a single-exponential algorithm?

We develop a new technique that allows us to reduce from non-disjunctive ASP to disjunctive ASP, thereby exponentially reducing structural dependencies. With this approach we trade a significantly simpler structure for additional computational hardness. Unfortunately, under ETH, we give negative answers by excluding most prominent structural parameters between vertex cover and treewidth: treedepth, feedback vertex size, cliquewidth, pathwidth, bandwidth, and cutwidth.

These results yield insights into the hardness of disjunction, thereby providing simple program structures that are already "hard". Indeed, from a practical point of view, this renders disjunction significantly harder, already for simple structural dependencies. In fact, disjunction allows us to simplify structure. So, how much does structure really help? We *hardly* expect better results for measures smaller than a vertex cover.

Future Work. This paper opens up several questions for future research. Combining structural and non-structural measures may be strong enough to better confine disjunction. This paper provides a starting point towards a modeling guideline for limiting the cost of disjunction. We expect progress in this direction in the future. Further, seeing how our bounds persist for stronger versions of ETH, e.g., SETH, is interesting. Do our reductions pay off in practice? Since the ones in Figures 3 and 5 preserve answer sets, we expect use cases for counting-like problems. For neuro-symbolic applications, where, (algebraic) counting is the basis for parameter learning, we expect improvements by our ideas. Here, as demonstrated by counting competitions (Fichte, Hecher, and Hamiti 2021; Korhonen and Järvisalo 2021), using structure is crucial.

Acknowledgments

Authors are ordered alphabetically. Part of the work has been carried out while Hecher visited the Simons Institute at UC Berkeley. Research is supported by the Austrian Science Fund (FWF), grants J4656 and P32830, and the Society for Research Funding in Lower Austria (GFF) grant ExzF-0004.

References

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

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

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

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

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

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

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

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

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

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

Calimeri, F.; Fuscà, D.; Perri, S.; and Zangari, J. 2016. I-dlv: The New Intelligent Grounder of dlv. In *AI*IA*, volume 10037 of *LNCS*, 192–207. Springer.

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

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

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

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

Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000. Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In *AAAI/IAAI'00*, 417–422. AAAI Press / The MIT Press.

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

Eiter, T.; Hecher, M.; and Kiesel, R. 2021. Treewidth-Aware Cycle Breaking for Algebraic Answer Set Counting. In *KR'21*, 269–279.

Fellows, M. R.; Rosamond, F. A.; Rotics, U.; and Szeider, S. 2009. Clique-Width is NP-Complete. *SIAM J. Discret. Math.*, 23(2): 909–939.

Fichte, J. K.; Ganian, R.; Hecher, M.; Slivovsky, F.; and Ordyniak, S. 2023. Structure-Aware Lower Bounds and Broadening the Horizon of Tractability for QBF. In *LICS'23*, 1–14.

Fichte, J. K.; Hecher, M.; and Hamiti, F. 2021. The Model Counting Competition 2020. *ACM J. Exp. Algorithmics*, 26.

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

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 *Theo. Comput. Science*. Springer. ISBN 978-3-540-29952-3.

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

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

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

Gottlob, G.; Pichler, R.; and Wei, F. 2010. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artificial Intelligence*, 174(1): 105–132.

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

Groenland, C.; Joret, G.; Nadara, W.; and Walczak, B. 2023. Approximating Pathwidth for Graphs of Small Treewidth. *ACM Trans. Algorithms*, 19(2): 16:1–16:19.

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

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

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

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

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

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

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

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

Kociumaka, T.; and Pilipczuk, M. 2014. Faster deterministic Feedback Vertex Set. *Inf. Process. Lett.*, 114(10): 556–560.

Korach, E.; and Solel, N. 1993. Tree-Width, Path-Widt, and Cutwidth. *Discret. Appl. Math.*, 43(1): 97–101.

Korhonen, T.; and Järvisalo, M. 2021. Integrating Tree Decompositions into Decision Heuristics of Propositional Model Counters. In *CP'21*, volume 210 of *LIPIcs*, 8:1–8:11. Dagstuhl Publishing.

Korhonen, T.; and Lokshtanov, D. 2023. An Improved Parameterized Algorithm for Treewidth. In *STOC'23*, 528–541. ACM.

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.

Lin, F.; and Zhao, J. 2003. On tight logic programs and yet another translation from normal logic programs to propositional logic. In *IJCAI'03*, 853–858. Morgan Kaufmann.

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

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

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

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

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

Niedermeier, R.; and Rossmanith, P. 1999. Upper bounds for vertex cover further improved. In *STACS 99: 16th Annual Symposium on Theoretical Aspects of Computer Science Trier, Germany, March 4–6, 1999 Proceedings 16*, 561–570. Springer.

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

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

Pan, G.; and Vardi, M. Y. 2006. Fixed-Parameter Hierarchies inside PSPACE. In *LICS'06*, 27–36. IEEE Computer Society.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison-Wesley. ISBN 0-470-86412-5.

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

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

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

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