Clique-Width and Directed Width Measures for Answer-Set Programming

Bernhard Bliem, Sebastian Ordyniak, and Stefan Woltran

TU Wien, Vienna, Austria

Abstract. Disjunctive Answer Set Programming (ASP) is a powerful declarative programming paradigm whose main decision problems are located on the second level of the polynomial hierarchy. Identifying tractable fragments and developing efficient algorithms for such fragments are thus important objectives in order to complement the sophisticated ASP systems available to date. Problems are fixed-parameter tractable (FPT) w.r.t. a parameter k if they can be solved in time $O(f(k) \cdot n^{O(1)})$ for some function f, where n is the input size. While several FPT results for ASP exist, parameters that relate to directed or signed graphs representing the program at hand have been neglected so far. In this paper, we first give some negative observations showing that directed width measures on the dependency graph of a program do not lead to FPT results. We then consider the graph parameter of signed clique-width and present a novel dynamic programming algorithm that is FPT w.r.t. this parameter. Clique-width is more general than the well-known treewidth, and, to the best of our knowledge, ours is the first FPT algorithm for bounded clique-width for reasoning problems beyond SAT.

1 Introduction

Disjunctive Answer Set Programming (ASP) [11, 27, 40] is an active field of AI providing a declarative formalism for solving hard computational problems. Thanks to the high sophistication of modern solvers [26], ASP was successfully used in several applications, including product configuration [47], decision support for space shuttle flight controllers [2], team scheduling [44], and bio-informatics [31].

Since the main decision problems of propositional ASP are located at the second level of the polynomial hierarchy [22, 49], the quest for easier fragments are important research contributions that could lead to improvements in ASP systems. An interesting approach to dealing with intractable problems comes from parameterized complexity theory [20] and is based on the fact that many hard problems become polynomial-time tractable if some problem parameter is bounded by a fixed constant. If the order of the polynomial bound on the runtime is independent of the parameter, one speaks of *fixed-parameter tractability* (FPT). Results in this direction for the ASP domain include [39] (parameter: size of answer sets), [38] (number of cycles), [5] (length of longest cycles), [4] (number of non-Horn rules), and [24] (backdoors). Also related is the parameterized complexity analysis of reasoning under subset-minimal models, see, e.g., [37].

As many prominent representations of logic programs are given in terms of directed graphs (consider, e.g., the dependency graph), it is natural to investigate parameters for ASP that apply to directed graphs. Over the past two decades, various width measures

for directed graphs have been introduced [35, 3, 6, 33, 45]. These are typically smaller than, e.g., the popular parameter of treewidth [8]. In particular, all these measures are zero on directed acyclic graphs (DAGs), but the treewidth of DAGs can be arbitrarily high. Moreover, since these measures are based on some notion of "closeness" to acyclicity and the complexity of ASP is closely related to the "cyclicity" of the rules in a program, such measures seem promising for obtaining algorithms that solve ASP in FPT time. Prominent applications of directed width measures include the k-Disjoint Path Problem [35], query evaluation in graph databases [1], and model checking [10].

Another graph parameter for capturing the structural complexity of a graph is clique-width [14, 16]. It applies to directed and undirected graphs, and in its general form (known as signed clique-width) to edge-labeled graphs. It is defined via graph construction where only a limited number of vertex labels is available; vertices that share the same label at a certain point of the construction process must be treated uniformly in subsequent steps. Constructions can be given by expressions in a graph grammar (so-called cwd-expressions) and the minimal number of labels required for constructing a graph G is the clique-width of G. While clique-width is in a certain way orthogonal to other directed width measures, it is more general than treewidth; there are classes of graphs with constant clique-width but arbitrarily high treewidth (e.g., complete graphs). In contrast, graphs with bounded treewidth also have bounded clique-width [16].

By means of a meta-theorem due to Courcelle, Makowsky, and Rotics [15], one can solve any graph problem that can be expressed in Monadic Second-Order Logic with quantification on vertex sets (MSO_1) in linear time for graphs of bounded clique-width. This result is similar to Courcelle's theorem [13] for graphs of bounded treewidth, which has been used for the FPT result for ASP w.r.t. treewidth [29]. There, the incidence graph of a program is used as an underlying graph structure (i.e., the graph containing a vertex for each atom a and rule r of the program, with an edge between a and r whenever a appears in r). Since the formula given in [29] is in MSO_1, the FPT result for ASP applies also to signed clique-width.

Clique-width is NP-hard to compute [23], which might be considered as an obstacle toward practical applications. However, one can check in polynomial time whether the width of a graph is bounded by a fixed k [43]. (These algorithms involve an additive approximation error that is bounded in terms of k). Recently, SAT solvers have been used to obtain sequences of vertex partitions that correspond to cwd-expressions [32] for a given graph. For some applications, it might not even be necessary to compute clique-width and the underlying cwd-expression: As mentioned in [25, Section 1.4], applications from the area of verification are supposed to already come with such an expression. Moreover, it might even be possible to partially obtain cwd-expressions during the grounding process of ASP.

This all calls for dedicated algorithms for solving ASP for programs of bounded clique-width. In contrast to treewidth where the FPT result from [29] has been used for designing [34] and implementing [41] a dynamic programming algorithm, to the best of our knowledge there are no algorithms yet that explicitly exploit the fixed-parameter tractability of ASP on bounded clique-width. In fact, we are not aware of any FPT algorithm for bounded clique-width for a reasoning problem located on the second level of the polynomial hierarchy (except [21] from the area of abstract argumentation).

The main contributions of this paper are as follows. First, we show some negative results for several *directed width measures*, indicating that the structure of the dependency graph and of various natural directed versions of the signed incidence graph does not adequately measure the complexity of evaluating the corresponding program.

Second, concerning *signed clique-width*, we give a novel dynamic programming algorithm that runs in polynomial time for programs where this parameter is bounded on their incidence graphs. We do so by suitably generalizing the seminal approach of [25] for the SAT problem. We also give a preliminary analysis how many signs are required in order to obtain FPT.

This work is an extended abstract of [7], which contains proofs that we omitted.

2 Preliminaries

Graphs We use standard graph terminology, see for instance the handbook [19]. All our graphs are simple. An undirected graph G is a tuple (V, E), where V or $V(G)$ is the vertex set and E or $E(G)$ is the edge set. For a subset $V' \subseteq V(G)$, we denote by $G[V']$, the *induced subgraph* of G induced by the vertices in V', i.e., $G[V']$ has vertices V' and edges $\{\, \{u, v\} \in E(G) \mid u, v \in V' \,\}$. We also denote by $G \setminus V'$ the graph $G[V(G) \setminus V']$. Similarly to undirected graphs, a digraph D is a tuple (V, A), where V or $V(D)$ is the vertex set and A or $A(D)$ is the *arc set*. A *strongly connected component* of a digraph D is a maximal subgraph Z of D that is strongly connected, i.e., Z contains a directed path between each pair of vertices in Z. We denote by $\mathrm{UND}(D)$ the *symmetric closure* of D, i.e., the graph with vertex set $V(D)$ and arc set $\{\, (u, v), (v, u) \mid (u, v) \in A(D) \,\}$. Finally, for a directed graph D, we denote by $\mathrm{DI}(G)$, the undirected graph with vertex set $V(G)$ and edge set $\{\, \{u, v\} \mid (u, v) \in A(D) \,\}$.

Parameterized Complexity In parameterized algorithmics [20] the runtime of an algorithm is studied with respect to a parameter $k \in \mathbb{N}$ and input size n. The most favorable class is FPT (*fixed-parameter tractable*) which contains all problems that can be decided by an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function. We also call such an algorithm fixed-parameter tractable, or FPT for short. Formally, a *parameterized problem* is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is the input alphabet. Let $L_1 \subseteq \Sigma_1^* \times \mathbb{N}$ and $L_2 \subseteq \Sigma_2^* \times \mathbb{N}$ be two parameterized problems. A *parameterized reduction* (or FPT-reduction) from L_1 to L_2 is a mapping $P : \Sigma_1^* \times \mathbb{N} \to \Sigma_2^* \times \mathbb{N}$ such that: (1) $(x, k) \in L_1$ iff $P(x, k) \in L_2$, (2) the mapping can be computed by an FPT-algorithm w.r.t. parameter k, and (3) there is a computable function g such that $k' \leq g(k)$, where $(x', k') = P(x, k)$. The class W[1] captures parameterized intractability and contains all problems that are FPT-reducible to PARTITIONED CLIQUE (the problem of deciding if a k-partite graph contains a clique of size k) when parameterized by the size of the solution. Showing W[1]-hardness for a problem rules out the existence of an FPT-algorithm under the usual assumption FPT $\neq$ W[1].

Answer Set Programming A *program* Π consists of a set $\mathcal{A}(\Pi)$ of propositional atoms and a set $\mathcal{R}(\Pi)$ of rules of the form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n,$$

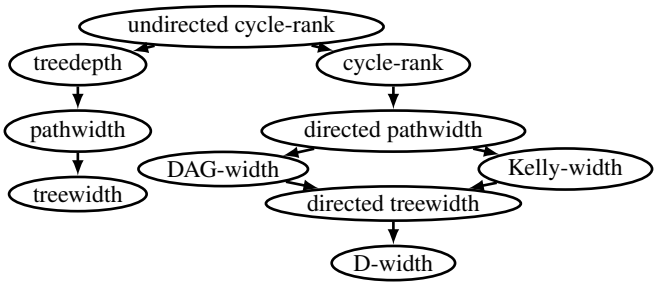

Fig. 1. Propagation of hardness results for the considered width measures. An arc (A, B) indicates that any hardness result parameterized by measure A implies a corresponding hardness result parameterized by B.

where $n \geq m \geq l$ and $a_i \in \mathcal{A}(\Pi)$ for $1 \leq i \leq n$. Each rule $r \in \mathcal{R}(\Pi)$ consists of a head $h(r) = \{a_1, \ldots, a_l\}$ and a body given by $p(r) = \{a_{l+1}, \ldots, a_m\}$ and $n(r) = \{a_{m+1}, \ldots, a_n\}$. A set $M \subseteq \mathcal{A}(\Pi)$ is a called a model of r if $p(r) \subseteq M$ and $n(r) \cap M = \emptyset$ imply $h(r) \cap M \neq \emptyset$. We denote the set of models of r by $\mathit{Mods}(r)$ and the models of Π are given by $\mathit{Mods}(\Pi) = \bigcap_{r \in \mathcal{R}(\Pi)} \mathit{Mods}(r)$.

The reduct Π^I of a program Π with respect to a set of atoms $I \subseteq \mathcal{A}(\Pi)$ is the program Π^I with $\mathcal{A}(\Pi^I) = \mathcal{A}(\Pi)$ and $\mathcal{R}(\Pi^I) = \{r^+ \mid r \in \mathcal{R}(\Pi),\ n(r) \cap I = \emptyset)\}$, where r^+ denotes rule r without negative body, i.e., $h(r^+) = h(r)$, $p(r^+) = p(r)$, and $n(r^+) = \emptyset$. Following [27], $M \subseteq \mathcal{A}(\Pi)$ is an *answer set* of Π if $M \in \mathit{Mods}(\Pi)$ and for no $N \subsetneq M$, we have $N \in \mathit{Mods}(\Pi^M)$. In what follows, we consider the problem of ASP consistency, i.e., the problem of deciding whether a given program has at least one answer set. As shown by Eiter and Gottlob, this problem is Σ_2^P-complete [22].

Graphical Representations of ASP Let Π be a program. The *dependency graph* of Π, denoted by DEP(Π), is the directed graph with vertex set $\mathcal{A}(\Pi)$ and that contains an arc (x, y) if there is a rule $r \in \mathcal{R}(\Pi)$ such that either $x \in h(r)$ and $y \in p(r) \cup n(r)$ or $x, y \in h(r)$ [24]. Note that there are other notions of dependency graphs used in the literature, most of them, however, are given as subgraphs of DEP(Π). As we will see later, our definition of dependency graphs allows us to draw immediate conclusions for such other notions.

The *incidence graph* of Π, denoted by INC(Π), is the undirected graph with vertices $\mathcal{A}(\Pi) \cup \mathcal{R}(\Pi)$ that contains an edge between a *rule vertex* $r \in \mathcal{R}(\Pi)$ and a *atom vertex* $a \in \mathcal{A}(\Pi)$ whenever $a \in h(r) \cup p(r) \cup n(r)$. The *signed incidence graph* of Π, denoted by SINC(Π), is the graph INC(Π), where addionally every edge of INC(Π) between an atom a and a rule r is annotated with a label from $\{h, p, n\}$ depending on whether a occurs in $h(r)$, $p(r)$, or $n(r)$.

3 Directed Width Measures

Since many representations of ASP programs are in terms of directed graphs, it is natural to consider parameters for ASP that are tailor-made for directed graphs. Over the past two decades various width measures for directed graphs have been introduced,

which are better suited for directed graphs than treewidth, on which they are based. The most prominent of those are directed treewidth [35], directed pathwidth [3], DAG-width [6], Kelly-width [33], and D-width [45] (see also [18]). Since these width measures are usually smaller on directed graphs than treewidth, it is worth considering them for problems that have already been shown to be fixed-parameter tractable parameterized by treewidth. In particular, all of these measure are zero on directed acyclic graphs (DAGs), but the treewidth of DAGs can be arbitrary high. Moreover, since these measures are based on some notion of "closeness" to acyclicity and the complexity of ASP is closely related to the "cyclicity" of the logical rules, one would consider such measures as promising for obtaining efficient algorithms for ASP.

In this section, we give results for directed width measures when applied to dependency graphs as defined in Section 2. To state our results in the most general manner, we will employ the parameter cycle-rank [12]. Since the cycle-rank is always greater or equal to any of the above mentioned directed width measures [30], any (parameterized) hardness result obtained for cycle-rank carries over to the aforementioned width measures for directed graphs.

Definition 1. *Let $D = (V, A)$ be a directed graph. The* cycle-rank *of D, denoted by* $\mathrm{cr}(D)$*, is inductively defined as follows: if D is acyclic, then* $\mathrm{cr}(D) = 0$*. Moreover, if D is strongly connected, then* $\mathrm{cr}(D) = 1 + \min_{v \in V} \mathrm{cr}(D \setminus \{v\})$*. Otherwise the cycle-rank of D is the maximum cycle-rank of any strongly connected component of D.*

For example, the cycle-rank of a disjoint union of cycles is 1, and the cycle-rank of a graph with an edge between each pair of vertices is equal to the number of vertices.

We will also consider a natural "undirected version" of the cycle-rank for directed graphs, i.e., we define the *undirected cycle-rank* of a directed graph D, denoted by $\mathrm{cr}^{\leftrightarrow}(D)$, to be the cycle-rank of $\mathrm{UND}(D)$. It is also well known (see, e.g., [28]) that the cycle-rank of $\mathrm{UND}(D)$ is equal to the treedepth of $\mathrm{DI}(D)$, i.e., the underlying undirected graph of D, and that the treedepth is always an upper bound for the pathwidth and the treewidth of an undirected graph [9]. Putting these facts together implies that any hardness result obtained for the undirected cycle-rank implies hardness for pathwidth, treewidth, treedepth as well as the aforementioned directed width measures. See also Figure 1 for an illustration how hardness results for the considered width measures propagate.

Finally, we would like to remark that both the cycle-rank and the undirected cycle-rank are easily seen to be closed under taking subgraphs, i.e., the (undirected) cycle-rank of a graph is always larger or equal to the (undirected) cycle-rank of every subgraph of the graph.

Hardness Results We show that ASP consistency remains as hard as in the general setting even for instances that have a dependency graph of constant width in terms of any of the directed width measures introduced.

For our hardness results, we employ the reduction given in [22] showing that ASP consistency is Σ_2^P-hard in general. The reduction is given from the validity problem for quantified Boolean formulas (QBF) of the form: $\Phi := \exists x_1 \cdots \exists x_n \forall y_1 \cdots \forall y_m \bigvee_{j=1}^{r} D_j$ where each D_j is a conjunction of at most three literals over the variables $x_1, \ldots, x_n$

$x_1 \vee v_1 \leftarrow$
$y_1 \vee z_1 \leftarrow$
$y_1 \leftarrow w$
$z_1 \leftarrow w$
$w \leftarrow y_1, z_1$
$w \leftarrow x_1, z_2$
$\leftarrow \neg w$

$x_2 \vee v_2 \leftarrow$
$y_2 \vee z_2 \leftarrow$
$y_2 \leftarrow w$
$z_2 \leftarrow w$
$w \leftarrow y_2, z_2$
$w \leftarrow v_2, y_2$

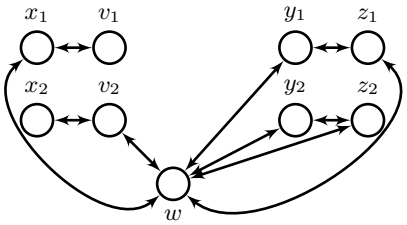

Fig. 2. Left: The program $\Pi(\Phi)$ for the formula $\Phi := \exists x_1 \exists x_2 \forall y_1 \forall y_2 (x_1 \wedge \neg y_2) \vee (\neg x_2 \wedge y_2)$. Right: The symmetric closure of the dependency graph of $\Pi(\Phi)$

and $y_1, \ldots, y_m$. We will denote the set of all QBF formulas of the above form in the following by $\text{QBF}_{2,\exists}^{\textbf{DNF}}$.

Given $\Phi \in \text{QBF}_{2,\exists}^{\textbf{DNF}}$, a program $\Pi(\Phi)$ is constructed as follows. The atoms of $\Pi(\Phi)$ are $x_1, v_1, \ldots, x_n, v_n, y_1, z_1, \ldots, y_m, z_m$, and w and $\Pi(\Phi)$ contains the following rules:

- for every i with $1 \leq i \leq n$, the rule $x_i \vee v_i \leftarrow$,
- for every i with $1 \leq i \leq m$, the rules $y_i \vee z_i \leftarrow$, $y_i \leftarrow w$, $z_i \leftarrow w$, and $w \leftarrow y_i, z_i$,
- for every j with $1 \leq j \leq r$, the rule $w \leftarrow \sigma(L_{j,1}), \sigma(L_{j,2}), \sigma(L_{j,3})$, where $L_{j,l}$ (for $l \in \{1,2,3\}$) is the l-th literal that occurs in D_j (if $|D_j| < 3$, the respective parts are omitted) and the function σ is defined by setting $\sigma(L)$ to v_i if $L = \neg x_i$, to z_i if $L = \neg y_i$, and to L otherwise.
- the rule $\leftarrow \neg w$ (i.e., with an empty disjunction in the head).

It has been shown [22, Theorem 38] that a $\text{QBF}_{2,\exists}^{\textbf{DNF}}$ formula Φ is valid iff $\Pi(\Phi)$ has an answer set. As checking validity of $\text{QBF}_{2,\exists}^{\textbf{DNF}}$ formulas is Σ_2^P-complete [48], this reduction shows that ASP is Σ_2^P-hard.

Lemma 1. *Let Φ be a $\text{QBF}_{2,\exists}^{\textbf{DNF}}$, then $\text{cr}^{\leftrightarrow}(\text{DEP}(\Pi(\Phi))) \leq 2$.*

Together with our considerations from above, we obtain:

Theorem 1. *ASP consistency is Σ_2^P-complete even for instances whose dependency graph has width at most two for any of the following width measures: undirected cycle-rank, pathwidth, treewidth, treedepth, cycle-rank, directed treewidth, directed path-width, DAG-width, Kelly-width, and D-width.*

Observe that because the undirected cycle-rank is closed under taking subgraphs and we chose the "richest" variant of the dependency graph, the above result carries over to the other notions of dependency graphs of ASP programs considered in the literature.

The above result draws a very negative picture of the complexity of ASP w.r.t. restrictions on the dependency graph. In particular, not even structural restrictions of the dependency graph by the usually very successful parameter treewidth can be employed for ASP. This is in contrast to our second graphical representation of ASP, the incidence graph, for which it is known that ASP is fixed-parameter tractable parameterized by the treewidth [34]. It is hence natural to ask whether the same still holds under restrictions provided by one of the directed width measures under consideration. We first need to discuss how to obtain a directed version of the usually undirected incidence graph.

For this, observe that the incidence graph, unlike the signed incidence graph, provides merely an incomplete model of the underlying ASP instance. Namely, it misses the information about *how* atoms occur in rules, i.e., whether they occur in the head, in the positive body, or in the negative body of a rule. A directed version of the incidence graph should therefore use the additional expressiveness provided by the direction of the arcs to incorporate the information given by the labels of the signed incidence graph. For instance, a natural directed version of the incidence graph could orient the edges depending on whether an atom occurs in the head or in the body of a rule. Clearly, there are many ways to orient the edges and it is not a priori clear which of those orientations leads to a directed version of the incidence graph that is best suited for an application of the directed width measures. Every orientation should, however, be consistent with the labels of the signed incidence graph, i.e., whenever two atoms are connected to a rule via edges having the same label, their arcs should be oriented in the same way. We call such an orientation of the incidence graph a *homogeneous* orientation.

Lemma 2. *Let Φ be a* $\mathrm{QBF}^{\boldsymbol{DNF}}_{2,\exists}$*, then the cycle-rank of any homogeneous orientation of the incidence graph of $\Pi(\Phi)$ is at most one.*

We can thus state the following result:

Theorem 2. *ASP consistency is Σ^P_2-complete even for instances whose directed incidence graph has width at most one for any of the following width measures: cycle-rank, directed treewidth, directed pathwidth, DAG-width, Kelly-width, and D-width.*

4 Clique-Width

The results in [29] imply that bounding the clique-width of the signed incidence graph of a program leads to tractability.

Proposition 1. *For a program Π such that the clique-width of its signed incidence graph is bounded by a constant, we can decide in linear time whether Π has an answer set.*

This result has been established via a formulation of ASP consistency as an MSO_1 formula. Formulating a problem in this logic automatically gives us an FPT algorithm. However, such algorithms are primarily of theoretical interest due to huge constant factors, and for actually solving problems, it is preferable to explicitly design dynamic programming algorithms [17].

Since our main tractability result considers the clique-width of an edge-labeled graph, i.e., the signed incidence graph, we will introduce clique-width for edge-labeled graphs. This definition also applies to graphs without edge-labels by considering all edges to be labeled with the same label. A *k-graph*, for $k > 0$, is a graph whose vertices are labeled by integers from $\{1, \ldots, k\} =: [k]$. Additionally, we also allow for the edges of a k-graph to be labeled by some arbitrary but finite set of labels (in our case the labels will correspond to the signs of the signed incidence graph). The labeling of the vertices of a graph $G = (V, E)$ is formally denoted by a function $\mathcal{L} : V \to [k]$. We consider an arbitrary graph as a k-graph with all vertices labeled by 1. We call the

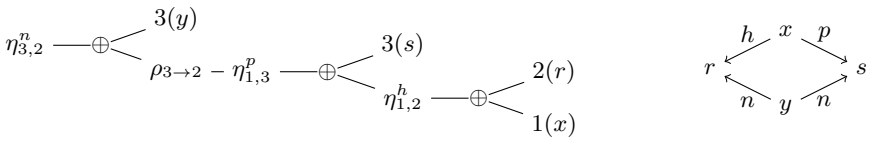

Fig. 3. A parse tree (left) of a 3-expression for SINC(Π) (right), where Π is the program consisting of the rules $x \leftarrow \neg y$ and $\leftarrow x, \neg y$

k-graph consisting of exactly one vertex v (say, labeled by $i \in [k]$) an *initial k-graph* and denote it by $i(v)$.

Graphs can be constructed from initial k-graphs by means of repeated application of the following three operations.

- *Disjoint union* (denoted by $\oplus$);
- *Relabeling*: changing all labels i to j (denoted by $\rho_{i\to j}$);
- *Edge insertion*: connecting all vertices labeled by i with all vertices labeled by j via an edge with label ℓ (denoted by $\eta^{\ell}_{i,j}$); $i \neq j$; already existing edges are not doubled.

A construction of a k-graph G using the above operations can be represented by an algebraic term composed of $i(v)$, $\oplus$, $\rho_{i\to j}$, and $\eta^{\ell}_{i,j}$, ($i, j \in [k]$, and v a vertex). Such a term is then called a *cwd-expression defining G*. For any cwd-expression σ, we use $\mathcal{L}_\sigma : V \to [k]$ to denote the labeling of the graph defined by σ. A *k-expression* is a cwd-expression in which at most k different labels occur. The set of all k-expressions is denoted by CW_k.

Definition 2. *The* clique-width *of a graph G,* cwd(G)*, is the smallest integer k such that G can be defined by a k-expression.*

For instance, trees have clique-width 3 and co-graphs have clique-width 2 (co-graphs are exactly given by the graphs which are P_4-free, i.e., whenever there is a path (a, b, c, d) in the graph then $\{a, c\}$, $\{a, d\}$ or $\{b, d\}$ is also an edge of the graph).

We have already introduced the notion of incidence graphs (resp. signed incidence graphs) of a program in Section 2. We thus can use cwd-expressions to represent programs.

Example 1. Let Π be the program with $\mathcal{A}(\Pi) = \{x, y\}$ and $\mathcal{R}(\Pi) = \{r, s\}$, where r is the rule $x \leftarrow \neg y$ and s is the rule $\leftarrow x, \neg y$. Its signed incidence graph SINC(Π) can be constructed by the 3-expression $\eta^n_{3,2}\Big(\rho_{3\to 2}\big(\eta^p_{1,3}(\eta^h_{1,2}(1(x)\oplus 2(r))\oplus 3(s))\big)\oplus 3(y)\Big)$, as depicted in Figure 3.

Since every k-expression of the signed incidence graph can be transformed into a k-expression of the unsigned incidence graph (by replacing all operations of the form $\eta^{\ell}_{i,j}$ with $\eta^{\alpha}_{i,j}$, where α is new label), it holds that cwd(INC(Π)) $\leq$ cwd(SINC(Π)).

Proposition 2. *Let Π be a program. It holds that* cwd(INC(Π)) $\leq$ cwd(SINC(Π))*, and there is a class $\mathcal{C}$ of programs such that, for each $\Pi \in \mathcal{C}$,* cwd(INC(Π)) $= 2$ *but* cwd(SINC(Π)) *is unbounded.*

For showing the second statement of the above proposition, consider a program Π_n that has n^2 atoms and n^2 rules (for some $n \in \mathbb{N}$), such that every atom occurs in every rule of Π_n. Because the incidence graph is a complete bipartite graph it has clique-width two and moreover it contains a grid G of size $n \times n$ as a subgraph. Assume that Π_n is defined in such a way that an atom a occurring in a rule r is in the head of r if the edge between a and r occurs in the grid G and otherwise a is in the (positive) body of r. Then, the clique-width of $\mathrm{SINC}(\Pi_n)$ is at least the clique-width of the $n \times n$ grid G, which grows with n [36]. Hence, the class $\mathcal{C}$ containing Π_n for every $n \in \mathbb{N}$ shows the second statement of the above proposition.

4.1 Algorithms

In this section, we provide our dynamic programming algorithms for deciding existence of an answer set. We start with the classical semantics for programs, where it is sufficient to just slightly adapt (a simplified version of) the algorithm for SAT by [25]. For answer-set semantics, we then extend this algorithm in order to deal with the intrinsic higher complexity of this semantics.

Both algorithms follow the same basic principles by making use of a k-expression σ defining a program Π via its signed incidence graph in the following way: We assign certain objects to each subexpression of σ and manipulate these objects in a bottom-up traversal of the parse tree of the k-expression such that the objects in the root of the parse tree then provide the necessary information to decide the problem under consideration. The size of these objects is bounded in terms of k (and independent of the size of Π) and the number of such objects required is linear in the size of Π. Most importantly, we will show that these objects can also be efficiently computed for bounded k. Thus, we will obtain the desired linear running time.

Classical Semantics

Definition 3. *A tuple $Q = (T, F, U)$ with $T, F, U \subseteq [k]$ is called a* k-triple*, and we refer to its parts using $Q_T = T$, $Q_F = F$, and $Q_U = U$. The set of all k-triples is given by $\mathcal{Q}_k$.*

The intuition of a triple (T, F, U) is to characterize a set of interpretations I in the following way:

- For each $i \in T$, at least one atom with label i is true in I;
- for each $i \in F$, at least one atom with label i is false in I;
- for each $i \in U$, there is at least one rule with label i that is "not satisfied yet".

Formally, the "semantics" of a k-triple Q with respect to a given program Π is given as follows.

Definition 4. *Let $Q \in \mathcal{Q}_k$ and Π be a program whose signed incidence graph (V, E) is labeled by $\mathcal{L} : V \to [k]$. A* Π-interpretation *of Q is a set $I \subseteq \mathcal{A}(\Pi)$ that satisfies*

$$Q_T = \{\mathcal{L}(a) \mid a \in I\},$$
$$Q_F = \{\mathcal{L}(a) \mid a \in \mathcal{A}(\Pi) \setminus I\}, \text{ and}$$
$$Q_U = \{\mathcal{L}(r) \mid r \in \mathcal{R}(\Pi),\ I \notin \mathit{Mods}(r)\}.$$

Example 2. Consider again program Π from Example 1 and the 3-expression σ from Figure 3. Let Q be the 3-triple $(\{1\},\{3\},\{2\})$. Observe that $\{x\}$ is a Π-interpretation of Q: It sets x to true and y to false, and $\mathcal{L}_\sigma(x) \in Q_T$ and $\mathcal{L}_\sigma(y) \in Q_F$ hold as required; the rule s is not satisfied by $\{x\}$, and indeed $\mathcal{L}_\sigma(s) \in Q_U$. We can easily verify that no other subset of $\mathcal{A}(\Pi)$ is a Π-interpretation of Q: Each Π-interpretation of Q must set x to true and y to false, as these are the only atoms labeled with 1 and 3, respectively.

We use the following notation for k-triples Q, Q', and set $S \subseteq [k]$.

- $Q \oplus Q' = (Q_T \cup Q'_T, Q_F \cup Q'_F, Q_U \cup Q'_U)$
- $Q^{i\to j} = (Q_T^{i\to j}, Q_F^{i\to j}, Q_U^{i\to j})$ where for $S \subseteq [k]$,

$$S^{i\to j} = S \setminus \{i\} \cup \{j\} \text{ if } i \in S \text{ and } S^{i\to j} = S \text{ otherwise.}$$

- $Q^{S,i,j} = (Q_T, Q_F, Q_U \setminus \{j\})$ if $i \in S$; $Q^{S,i,j} = Q$ otherwise.

Using these abbreviations, we define our dynamic programming algorithm: We assign to each subexpression σ of a given k-expression a set of triples by recursively defining a function f, which associates to σ a set of k-triples as follows.

Definition 5. *The function $f : CW_k \to 2^{\mathcal{Q}_k}$ is recursively defined along the structure of k-expressions as follows.*

- $f(i(v)) = \begin{cases} \big\{\big(\{i\},\emptyset,\emptyset\big),\ \big(\emptyset,\{i\},\emptyset\big)\big\} & \text{if } v \text{ is an atom node} \\ \big\{\big(\emptyset,\emptyset,\{i\}\big)\big\} & \text{if } v \text{ is a rule node} \end{cases}$
- $f(\sigma_1 \oplus \sigma_2) = \{Q \oplus Q' \mid Q \in f(\sigma_1), Q' \in f(\sigma_2)\}$
- $f(\rho_{i\to j}(\sigma)) = \{Q^{i\to j} \mid Q \in f(\sigma)\}$
- $f(\eta^h_{i,j}(\sigma)) = f(\eta^n_{i,j}(\sigma)) = \{Q^{Q_T,i,j} \mid Q \in f(\sigma)\}$
- $f(\eta^p_{i,j}(\sigma)) = \{Q^{Q_F,i,j} \mid Q \in f(\sigma)\}$

Example 3. Consider again program Π from Example 1 and the 3-expression depicted in Figure 3. To break down the structure of σ, let $\sigma_1, \ldots, \sigma_6$ be subexpressions of σ such that $\sigma = \eta^n_{3,2}(\sigma_1)$, $\sigma_1 = \sigma_2 \oplus 3(y)$, $\sigma_2 = \rho_{3\to 2}(\sigma_3)$, $\sigma_3 = \eta^p_{1,3}(\sigma_4)$, $\sigma_4 = \sigma_5 \oplus 3(s)$, $\sigma_5 = \eta^h_{1,2}(\sigma_6)$ and $\sigma_6 = 1(x) \oplus 2(r)$. We get $f(1(x)) = \big\{(\{1\},\emptyset,\emptyset),(\emptyset,\{1\},\emptyset)\big\}$ and $f(2(r)) = \big\{(\emptyset,\emptyset,\{2\})\big\}$. These sets are then combined to $f(\sigma_6) = \big\{(\{1\},\emptyset,\{2\}),$ $(\emptyset,\{1\},\{2\})\big\}$. The program defined by σ_6 consists of atom x and rule r, but x does not occur in r yet. Accordingly, the k-triple $(\{1\},\emptyset,\{2\})$ models the situation where x is set to true, which does not satisfy r (since the head and body of r are still empty), hence the label of r is in the last component; the 3-triple $(\emptyset,\{1\},\{2\})$ represents x being set to false, which does not satisfy r either. Next, σ_5 causes all atoms with label 1 (i.e., just x) to be inserted into the head of all rules with label 2 (i.e., just r), and we get $f(\sigma_5) = \big\{(\{1\},\emptyset,\emptyset),(\emptyset,\{1\},\{2\})\big\}$. We obtain the first element $(\{1\},\emptyset,\emptyset) = Q^{Q_T,1,2}$ from $Q = (\{1\},\emptyset,\{2\})$ by removing the label 2 from Q_U because $1 \in Q_T$. The idea is that the heads of all rules labeled with 2 now contain all atoms labeled with 1, so these rules become satisfied by every interpretation that sets some atom labeled with 1 to true. Next, σ_4 adds the rule s with label 3 and we get $f(\sigma_4) = \big\{(\{1\},\emptyset,\{3\}),(\emptyset,\{1\},\{2,3\})\big\}$. The edge added by σ_3 adds all atoms with label 1

(i.e., just x) into the positive body of all rules with label 3 (i.e., just s), which results in $f(\sigma_3) = \big\{(\{1\}, \emptyset, \{3\}), (\emptyset, \{1\}, \{2\})\big\}$. Observe that the last component of the second element no longer contains 3, i.e., setting x to false makes s true. Now the label 3 is renamed to 2, and we get $f(\sigma_2) = \big\{(\{1\}, \emptyset, \{2\}), (\emptyset, \{1\}, \{2\})\big\}$. Note that now r and s are no longer distinguishable since they now share the same label. Hence all operations that add edges to r will also add edges to s and vice versa. In σ_1, atom y is added with label 3 and we get four 3-triples in $f(\sigma_1)$: From $(\{1\}, \emptyset, \{2\})$ in $f(\sigma_2)$ we obtain $(\{1, 3\}, \emptyset, \{2\})$ and $(\{1\}, \{3\}, \{2\})$, and from $(\emptyset, \{1\}, \{2\})$ in $f(\sigma_2)$ we get $(\{3\}, \{1\}, \{2\})$ and $(\emptyset, \{1, 3\}, \{2\})$. In σ, we add a negative edge from all atoms labeled with 3 (i.e., just y) to all rules labeled with 2 (both r and s). From $(\{1, 3\}, \emptyset, \{2\})$ in $f(\sigma_1)$ we now get $(\{1, 3\}, \emptyset, \emptyset)$, from $(\{3\}, \{1\}, \{2\})$ we get $(\{3\}, \{1\}, \emptyset)$, and the 3-triples $(\{1\}, \{3\}, \{2\})$ and $(\emptyset, \{1, 3\}, \{2\})$ from $f(\sigma_1)$ occur unmodified in $f(\sigma)$. As we will prove shortly, for each k-triple Q in $f(\sigma)$, there is a Π-interpretation of Q. So if there is a k-triple Q in $f(\sigma)$ such that $Q_U = \emptyset$, then Π has a classical model due to the definition of Q_U. For instance, $(\{1, 3\}, \emptyset, \emptyset)$ has a Π-interpretation $\{x, y\}$, which is obviously a model of Π.

Lemma 3. *Let Π be a program and θ be a k-expression for* SINC(Π). *For every set $I \subseteq \mathcal{A}(\Pi)$, there is a k-triple $Q \in f(\theta)$ such that I is a Π-interpretation of Q, and for every k-triple $Q \in f(\theta)$ there is a set $I \subseteq \mathcal{A}(\Pi)$ such that I is a Π-interpretation of Q.*

Theorem 3. *Let k be an integer and Π be a program. Given a k-expression for the signed incidence graph of Π, we can decide in linear time whether Π has a model.*

Answer-Set Semantics For full disjunctive ASP we need a more involved data structure.

Definition 6. *A pair (Q, Γ) with with $Q \in \mathcal{Q}_k$ and $\Gamma \subseteq \mathcal{Q}_k$ is called a* k-pair. *The set of all k-pairs is given by $\mathcal{P}_k$.*

Given a k-pair (Q, Γ), the purpose of Q is, as for classical semantics, to represent Π-interpretations I (that in the end correspond to models). Every k-triple in Γ represents sets J of atoms such that $J \subset I$. If, in the end, there is such a set J that still satisfies every rule in the reduct w.r.t. I, then we conclude that I is not an answer set.

Definition 7. *Let $Q \in \mathcal{Q}_k$, let Π be a program whose signed incidence graph (V, E) is labeled by $\mathcal{L} : V \to [k]$, and let $I \subseteq \mathcal{A}(\Pi)$. A* Π^I-interpretation *of Q is a set $J \subseteq \mathcal{A}(\Pi)$ such that*

$$
\begin{aligned}
&Q_T = \{\mathcal{L}(a) \mid a \in J\},\\
&Q_F = \{\mathcal{L}(a) \mid a \in \mathcal{A}(\Pi) \setminus J\}, \text{ and}\\
&Q_U = \{\mathcal{L}(r) \mid r \in \mathcal{R}(\Pi),\ n(r) \cap I = \emptyset,\ J \notin \mathit{Mods}(r^+)\}.
\end{aligned}
$$

We can now define our dynamic programming algorithm for ASP:

Definition 8. *The function $g : CW_k \to 2^{\mathcal{P}_k}$ is recursively defined along the structure of k-expressions as follows.*

- $g(i(v)) = \Big\{\big((\{i\},\emptyset,\emptyset),\ \{(\emptyset,\{i\},\emptyset)\}\big),\quad \big((\emptyset,\{i\},\emptyset),\ \emptyset\big)\Big\}$ *if v is at atom node*
- $g(i(v)) = \Big\{\big((\emptyset,\emptyset,\{i\}),\ \emptyset\big)\Big\}$ *if v is a rule node*
- $g(\sigma_1 \oplus \sigma_2) = \{\big(Q_1 \oplus Q_2, R_{Q_1,Q_2,\Gamma_1,\Gamma_2} \mid (Q_i,\Gamma_i) \in g(\sigma_i)\big)\}$, *where* $R_{Q_1,Q_2,\Gamma_1,\Gamma_2} = \big\{S_1 \oplus S_2 \mid S_i \in \Gamma_i\big\} \cup \big\{Q_1 \oplus S \mid S \in \Gamma_2\big\} \cup \big\{S \oplus Q_2 \mid S \in \Gamma_1\big\}\big)$
- $g(\rho_{i\to j}(\sigma)) = \{\big((Q^{i\to j}, \{R^{i\to j} \mid R \in \Gamma\}\big) \mid (Q,\Gamma) \in g(\sigma)\}$
- $g(\eta^h_{i,j}(\sigma)) = \{\big(Q^{Q_T,i,j}, \{R^{R_T,i,j} \mid R \in \Gamma\}\big) \mid (Q,\Gamma) \in g(\sigma)\}$
- $g(\eta^p_{i,j}(\sigma)) = \{\big(Q^{Q_F,i,j}, \{R^{R_F,i,j} \mid R \in \Gamma\}\big) \mid (Q,\Gamma) \in g(\sigma)\}$
- $g(\eta^n_{i,j}(\sigma)) = \{\big(Q^{Q_T,i,j}, \{R^{Q_T,i,j} \mid R \in \Gamma\}\big) \mid (Q,\Gamma) \in g(\sigma)\}$

Note the use of Q_T in $R^{Q_T,i,j}$ in the definition of $g(\eta^n_{i,j}(\sigma))$: Whenever an interpretation I represented by Q sets an atom from the negative body of a rule r to true, the rule r has no counterpart in the reduct w.r.t. I, so, for each subset J of I, we remove r from the set of rules whose counterpart in the reduct is not yet satisfied by J.

Example 4. Let Π be the program consisting of a single rule $\leftarrow \neg x$, which we denote by r, and let $\sigma = \eta^n_{1,2}(1(x) \oplus 2(r))$. Let (Q,Γ) be the k-pair in $g(1(x))$ with $Q = (\{1\},\emptyset,\emptyset)$ and $\Gamma = \{(\emptyset,\{1\},\emptyset)\}$. The k-triple Q represents the set of atoms $\{x\}$. Since this set has the proper subset $\emptyset$, there is a k-triple in Γ that indeed corresponds to this subset. Now let $(Q,\Gamma) = ((\emptyset,\{1\},\emptyset),\ \emptyset)$ be the other k-pair in $g(1(x))$. Here Q represents the empty set of atoms, which has no proper subsets, hence Γ is empty. For the single k-pair $((\emptyset,\emptyset,\{2\}),\ \emptyset)$ in $g(2(r))$, the situation is similar. Next, at $g(1(x) \oplus 2(r))$, we combine every k-pair (Q_1,Γ_1) from $g(1(x))$ with every k-pair (Q_2,Γ_2) from $g(2(r))$ to a new k-pair. For instance, consider $Q_1 = (\{1\},\emptyset,\emptyset)$ and $\Gamma_1 = \{S\}$, where $S = (\emptyset,\{1\},\emptyset)$, as well as $Q_2 = (\emptyset,\emptyset,\{2\})$ and $\Gamma_2 = \emptyset$. By definition of g, we obtain a new k-pair (Q,Γ), where $Q = Q_1 \oplus Q_2 = (\{1\},\emptyset,\{2\})$, and Γ contains the single element $Q_2 \oplus S = (\emptyset,\{1\},\{2\})$. Recall that the purpose of Q is to represent sets of atoms I, and each element of Γ shall represent proper subsets of I; in this case, Q represents $\{x\}$, and the element $Q_2 \oplus S$ in Γ represents the proper subset $\emptyset$. Next, at $g(\sigma)$ we introduce a negative edge from x to r. From the k-pair $(Q,\{S\})$ in $g(1(x) \oplus 2(r))$, where $Q = (\{1\},\emptyset,\{2\})$ and $S = (\emptyset,\{1\},\{2\})$, we obtain the k-pair $(Q',\{S'\})$ in $g(\sigma)$, where $Q' = Q^{Q_T,i,j} = (\{1\},\emptyset,\emptyset)$ (i.e., the label 2 from Q_U has disappeared) and $S' = S^{Q_T,i,j} = (\emptyset,\{1\},\emptyset)$. Here 2 has disappeared from S_U because the reduct w.r.t. all sets of atoms represented by Q' no longer contains any rule labeled with 2. Note that the classical model $\{x\}$ represented by Q' is no answer set even though $Q'_U = \emptyset$. The reason is that S' witnesses (by $S'_U = \emptyset$) that $\emptyset \in \mathit{Mods}(\Pi^{\{x\}})$.

Lemma 4. *Let Π be a program and θ be a k-expression for* SINC(Π). *For every set $I \subseteq \mathcal{A}(\Pi)$ there is a k-pair $(Q,\Gamma) \in g(\theta)$ such that (i) I is a Π-interpretation of Q and (ii) for every set $J \subset I$ there is a k-triple $R \in \Gamma$ such that J is a Π^I-interpretation of R. Moreover, for every k-pair $(Q,\Gamma) \in g(\theta)$ there is a set $I \subseteq \mathcal{A}(\Pi)$ such that (i') I is a Π-interpretation of Q and (ii') for each k-triple $R \in \Gamma$, there is a set $J \subset I$ such that J is a Π^I-interpretation of R.*

Theorem 4. *Let k be a constant and Π be a program. Given a k-expression for* SINC(Π), *we can decide in linear time whether Π has an answer set.*

4.2 The Role of Signs for Results on Clique-Width

As we have seen, ASP parameterized by the clique-width of the signed incidence graph is FPT. As the clique-width of the (unsigned) incidence graph is at most the clique-width of the signed incidence graph (Proposition 2), an FPT result w.r.t. the clique-width of the (unsigned) incidence graph would be significantly stronger. The propositional satisfiability problem (SAT) was shown in [25] to be FPT parameterized by the clique-width of the signed incidence graph, and the authors conjectured that the same should hold for the unsigned version. Surprisingly, it turned out not to be the case [42]. The situation for ASP is slightly more involved. While there are only two signs for SAT (i.e., whether a variable occurs positively or negatively in a clause), ASP has three signs (h, p, n). So how many signs are necessary to obtain tractability for ASP? For this, let $\mathrm{SINC}_L(\Pi)$, for $L \subseteq \{h, p, n\}$, be the ("semi-signed") incidence graph obtained from $\mathrm{SINC}(\Pi)$ by joining all labels in L, i.e., every label in L is renamed to a new label α. We show that joining any set L of labels other than $\{h, n\}$ leads to intractability for ASP parameterized by the clique-width of $\mathrm{SINC}_L(\Pi)$. Together with our tractability result w.r.t. the clique-width of $\mathrm{SINC}(\Pi)$ (Theorem 4), this provides an almost complete picture of the complexity of ASP parameterized by clique-width. We leave it open whether ASP parameterized by the clique-width of $\mathrm{SINC}_{\{h,n\}}(\Pi)$ is FPT.

Theorem 5. *Let $L \subseteq \{h, p, n\}$ with $|L| > 1$ and $L \neq \{h, n\}$, then ASP is* W[1]*-hard parameterized by the clique-width of* $\mathrm{SINC}_L(\Pi)$.

5 Conclusion

In this paper, we contributed to the parameterized complexity analysis of ASP. We first gave negative results: Most directed width measures (applied to the dependency graph or incidence graph of a program) do not lead to FPT. Then we turned a theoretical tractability result (which follows from [29]) for the parameter clique-width (of the signed incidence graph of a program) into a novel dynamic programming algorithm, which is applicable to arbitrary programs when given a k-expression. It is expected to be efficient for small k, i.e., programs whose signed incidence graph has low clique-width.

Beside studying different parameters (e.g., rank-width), future work includes the complexity of ASP parameterized by the clique-width of $\mathrm{SINC}_{\{h,n\}}(\Pi)$ and of the unsigned incidence graph (where SAT is in the class XP [46]). Another topic is which classes of non-ground ASP programs preserve bounded clique-width of the input.

Acknowledgments. This work was supported by the Austrian Science Fund (FWF) projects P25518 and Y698.

References

1. Bagan, G., Bonifati, A., Groz, B.: A trichotomy for regular simple path queries on graphs. In: Proc. PODS. pp. 261–272. ACM (2013)
2. Balduccini, M., Gelfond, M., Nogueira, M.: Answer set based design of knowledge systems. Ann. Math. Artif. Intell. 47(1-2), 183–219 (2006)

3. Barát, J.: Directed path-width and monotonicity in digraph searching. Graphs and Combinatorics 22(2), 161–172 (2006)
4. Ben-Eliyahu, R.: A hierarchy of tractable subsets for computing stable models. J. Artif. Intell. Res. (JAIR) 5, 27–52 (1996)
5. Ben-Eliyahu, R., Dechter, R.: Propositional semantics for disjunctive logic programs. Ann. Math. Artif. Intell. 12(1-2), 53–87 (1994)
6. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S., Obdrzálek, J.: The dag-width of directed graphs. Journal of Combinatorial Theory, Series B 102(4), 900–923 (2012)
7. Bliem, B., Ordyniak, S., Woltran, S.: Clique-width and directed width measures for answer-set programming. CoRR abs/1606.09449 (2016), http://arxiv.org/abs/1606.09449
8. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11, 1–21 (1993)
9. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J. Algorithms 18(2), 238–255 (1995)
10. Bojańczyk, M., Dittmann, C., Kreutzer, S.: Decomposition theorems and model-checking for the modal μ-calculus. In: Proc. CLS/LICS. pp. 17:1–17:10. ACM (2014)
11. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Comm. ACM 54(12), 92–103 (2011)
12. Cohen, R.S.: Transition graphs and the star height problem. In: Proc. of the 9th Annual Symposium on Switching and Automata Theory. pp. 383–394. IEEE Computer Society (1968)
13. Courcelle, B.: Recognizability and second-order definability for sets of finite graphs. Tech. Rep. I-8634, Université de Bordeaux (1987)
14. Courcelle, B., Engelfriet, J., Rozenberg, G.: Context-free handle-rewriting hypergraph grammars. In: Proc. Graph-Grammars. LNCS, vol. 532, pp. 253–268 (1991)
15. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33(2), 125–150 (2000)
16. Courcelle, B., Olariu, S.: Upper bounds to the clique-width of graphs. Discr. Appl. Math. 101(1-3), 77–114 (2000)
17. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)
18. Dehmer, M., Emmert-Streib, F. (eds.): Quantitative Graph Theory, chap. Width-Measures for Directed Graphs and Algorithmic Applications. CRC Press (2014)
19. Diestel, R.: Graph Theory, 4th Edition, Graduate texts in mathematics, vol. 173. Springer (2012)
20. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science, Springer (2013)
21. Dvorák, W., Szeider, S., Woltran, S.: Reasoning in argumentation frameworks of bounded clique-width. In: Proc. COMMA. Frontiers in Artificial Intelligence and Applications, vol. 216, pp. 219–230. IOS Press (2010)
22. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell. 15(3–4), 289–323 (1995)
23. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width is NP-complete. SIAM J. Discrete Math. 23(2), 909–939 (2009)
24. Fichte, J., Szeider, S.: Backdoors to tractable answer set programming. Artif. Intell. 220, 64–103 (2015)
25. Fischer, E., Makowsky, J.A., Ravve, E.R.: Counting truth assignments of formulas of bounded tree-width or clique-width. Discr. Appl. Math. 156(4), 511–529 (2008)
26. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)

27. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
28. Giannopoulou, A.C., Hunter, P., Thilikos, D.M.: LIFO-search: A min-max theorem and a searching game for cycle-rank and tree-depth. Discrete Applied Mathematics 160(15), 2089–2097 (2012)
29. Gottlob, G., Pichler, R., Wei, F.: Bounded treewidth as a key to tractability of knowledge representation and reasoning. Artif. Intell. 174(1), 105–132 (2010)
30. Gruber, H.: Digraph complexity measures and applications in formal language theory. Discrete Mathematics & Theoretical Computer Science 14(2), 189–204 (2012)
31. Guziolowski, C., Videla, S., Eduati, F., Thiele, S., Cokelaer, T., Siegel, A., Saez-Rodriguez, J.: Exhaustively characterizing feasible logic models of a signaling network using answer set programming. Bioinformatics 29(18), 2320–2326 (2013), erratum see Bioinformatics 30, 13, 1942.
32. Heule, M., Szeider, S.: A SAT approach to clique-width. ACM Trans. Comput. Log. 16(3), 24 (2015)
33. Hunter, P., Kreutzer, S.: Digraph measures: Kelly decompositions, games, and orderings. TCS 399(3), 206–219 (2008)
34. Jakl, M., Pichler, R., Woltran, S.: Answer-set programming with bounded treewidth. In: Proc. IJCAI. pp. 816–822 (2009)
35. Johnson, T., Robertson, N., Seymour, P.D., Thomas, R.: Directed tree-width. Journal of Combinatorial Theory, Series B 82(1), 138–154 (2001)
36. Kaminski, M., Lozin, V.V., Milanic, M.: Recent developments on graphs of bounded clique-width. Discrete Applied Mathematics 157(12), 2747–2761 (2009)
37. Lackner, M., Pfandler, A.: Fixed-parameter algorithms for finding minimal models. In: Proc. KR. pp. 85–95. AAAI Press (2012)
38. Lin, F., Zhao, X.: On odd and even cycles in normal logic programs. In: Proc. AAAI. pp. 80–85. AAAI Press / The MIT Press (2004)
39. Lonc, Z., Truszczyński, M.: Fixed-parameter complexity of semantics for logic programs. ACM Trans. Comput. Log. 4(1), 91–119 (2003)
40. Marek, V.W., Truszczyński, M.: Stable Models and an Alternative Logic Programming Paradigm. In: The Logic Programming Paradigm – A 25-Year Perspective, pp. 375–398. Springer Verlag (1999)
41. Morak, M., Pichler, R., Rümmele, S., Woltran, S.: A dynamic-programming based ASP-solver. In: Proc. JELIA'10. pp. 369–372 (2010)
42. Ordyniak, S., Paulusma, D., Szeider, S.: Satisfiability of acyclic and almost acyclic CNF formulas. TCS 481, 85–99 (2013)
43. Oum, S., Seymour, P.: Approximating clique-width and branch-width. J. Combin. Theory Ser. B 96(4), 514–528 (2006)
44. Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-building with answer set programming in the Gioia-Tauro seaport. TPLP 12, 361–381 (4 2012)
45. Safari, M.A.: D-width: A more natural measure for directed tree width. In: Proc. MFCS. LNCS, vol. 3618, pp. 745–756. Springer (2005)
46. Slivovsky, F., Szeider, S.: Model counting for formulas of bounded clique-width. In: Proc. ISAAC. LNCS, vol. 8283, pp. 677–687. Springer (2013)
47. Soininen, T., Niemelä, I.: Developing a declarative rule language for applications in product configuration. In: Proc. PADL. LNCS, vol. 1551, pp. 305–319. Springer Verlag (1998)
48. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proc. Theory of Computing. pp. 1–9. ACM (1973)
49. Truszczyński, M.: Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs. TPLP 11(6), 881–904 (2011)