The Impact of Treewidth on ASP Grounding and Solving

Bernhard Bliem, Marius Moldovan, Michael Morak, and **Stefan Woltran**

TU Wien, Vienna, Austria

{bliem,moldovan,morak,woltran}@dbai.tuwien.ac.at

Abstract

In this paper, we aim to study how the performance of modern answer set programming (ASP) solvers is influenced by the treewidth of the input program and to investigate the consequences of this relationship. We first perform an experimental evaluation that shows that the solving performance is heavily influenced by the treewidth, given ground input programs that are otherwise uniform, both in size and construction. This observation leads to an important question for ASP, namely, how to design encodings such that the treewidth of the resulting ground program remains small. To this end, we define the class of connection-guarded programs, which guarantees that the treewidth of the program after grounding only depends on the treewidth (and the degree) of the input instance. In order to obtain this result, we formalize the grounding process using MSO transductions.

1 Introduction

Answer set programming (ASP) [Marek and Truszczyński, 1999; Brewka *et al.*, 2011; Gebser *et al.*, 2012a] is a well-established logic programming paradigm based on the stable model semantics. Its main benefit is an intuitive, declarative language, and the fact that, generally, each answer set of a given logic program describes a valid solution of the original problem. Solving ASP programs is usually a two-step process. First, a (usually fixed) encoding for a given problem is written in the language of non-ground ASP. This encoding, together with a set of input facts representing the actual problem instance, gets passed to a *grounder* which transforms it into an equivalent propositional ASP program. In the second step, this ground program is then evaluated by a *solver*. Such ASP solvers are now readily available (e.g., [Gebser *et al.*, 2012b; Alviano *et al.*, 2015; Elkabani *et al.*, 2005; Leone *et al.*, 2006]) and have made huge strides in efficiency.

This leads to the following interesting practical question: What is the relationship of solver efficiency and different parameters of the ground input program, and how is the solving time influenced by these parameters? On the theoretical side, computational complexity investigations were carried out for the classical parameter of input size [Eiter and Gottlob, 1995; Truszczyński, 2011; Dantsin *et al.*, 2001], while several structural parameters were studied in the field of parameterized complexity [Gottlob *et al.*, 2010; Pichler *et al.*, 2014; Fichte and Szeider, 2015]. This has also led to specialized implementations that try to explicitly exploit these parameters [Jakl *et al.*, 2009; Fichte *et al.*, 2017]. While these theoretical investigations provide us with valuable insight into the problem of ASP solving, it is not obvious what conclusions can be drawn for the actual practical solving performance of today's top-of-the-line ASP solvers. It would be interesting to see how current CDCL-based solvers are influenced, in practice, by variations in such structural parameters and whether guidelines for ASP modeling can be derived from such interactions. One of the few results in this direction is the discovery of a strong correlation between the rule-to-atom ratio of a ground ASP program and the solving time [Zhao and Lin, 2003], a property that carries over from similar studies for SAT [Selman *et al.*, 1996]. A more recent study on phase transitions in ASP also deals with this topic [Wen *et al.*, 2016]. Beside these results, however, the practical impact of structural parameters on solving time has not, in the authors' opinion, received adequate attention in the literature.

In this paper we focus on the parameter of treewidth, a measure of how closely a ground ASP program structurally resembles a tree. Our goal is to study how the performance of modern ASP solvers is influenced by the treewidth of the given ground input program and to investigate the consequences of this relationship. To this end, our first main contribution is to carry out an extensive experimental evaluation of two top-of-the-line ASP solver implementations and investigate how the solving performance behaves when the solvers are presented with hard instances of uniform size and construction, but variable treewidth, using a carefully crafted ASP problem encoding and adequately generated, tree-like instances. Our experiments show that the solving time for programs of the same size and construction indeed increases drastically with the treewidth. This is an interesting result, which shows that similar results for SAT solvers and resolution-width [Atserias *et al.*, 2011] do indeed carry over to the more complex world of ASP. Our observations suggest that, when encoding problems in (non-ground) ASP, it is not only important that the resulting ground program is small, but also that its treewidth is kept as small as possible, given a set of input facts.

Example 1. Reachability can be modeled in different ways using ASP. One way would be to model the transitive closure of a graph as follows (where `e` is the predicate representing graph edges and `r` the predicate to mark reachable vertices):

```
t(X,Y) :- e(X,Y).
t(X,Z) :- t(X,Y), e(Y,Z).
r(Y)   :- t(X,Y), start(X).
```

However, when used as a sub-program that the grounder has to instantiate, such an encoding causes any two (connected) vertices in the input graph to appear together in a rule after grounding (in place of the variables `X` and `Z` in the second rule). This then causes the graph representation of the ground program to contain a clique whose size equals the number of vertices of the original input graph, resulting in a high treewidth. Conventional wisdom in ASP would recommend the following encoding:

```
r(X) :- start(X).
r(Y) :- e(X,Y), r(X).
```

Here, not only is the grounding smaller, but also the treewidth decreases dramatically. In fact, it now solely depends on the treewidth (and not the size) of the input graph. □

This example illustrates that the way a problem is encoded can influence the treewidth of the ground program considerably. Due to the grounding step, however, it is not obvious at the time of writing a non-ground ASP encoding how to achieve a low-treewidth grounding and the benefits that come with it. This is as opposed to, for example, SAT formulas that can be generated directly while keeping treewidth in mind. The second main contribution of this paper addresses this issue and allows us to leverage the treewidth-sensitivity of ASP solvers: We define the class of *connection-guarded* programs, which guarantees that the treewidth of a program after grounding does not increase arbitrarily but only depends on the treewidth (and degree) of the input facts. We also show that programs in our class are, at the same time, expressive enough to encode relevant problems from the second level of the polynomial hierarchy. In our proofs, we use the notion of MSO transductions [Courcelle and Engelfriet, 2012] to formally represent the grounding process and investigate its influence on the treewidth. To our knowledge, this is the first time that this technique has been used in the context of ASP.

The remainder of the paper is structured as follows. In Section 2, we give relevant definitions for ASP, treewidth, and MSO transductions. Section 3 deals with our first main contribution, the experimental evaluation of solver performance with respect to treewidth, and shows that there is a significant correlation. Section 4 presents our second main contribution, namely, proposing the class of connection-guarded ASP programs that aims to preserve the treewidth of the given instance after grounding. We also address implications of our results for ASP solving and modeling. Finally, we conclude the paper with a discussion in Section 5.

2 Preliminaries

Answer Set Programming (ASP). ASP is a declarative problem modeling and solving framework with a complex lan-

guage that we only briefly introduce here. For a full, formal introduction, we refer to other sources [Brewka *et al.*, 2011; Gebser *et al.*, 2012b]. A *non-ground disjunctive logic program* Π consists of a set of rules of the form $r : \underline{h_1} \vee \cdots \vee \underline{h_k} \leftarrow \underline{p_1}, \ldots, \underline{p_n}, \neg \underline{n_1}, \ldots, \neg \underline{n_m}$, where $\underline{h_i}$, $\underline{p_i}$ and $\underline{n_i}$ are *atoms*, called head ($H(r)$), positive and negative body atoms ($B^+(r)$ and $B^-(r)$), respectively. An atom $\underline{a}$ is of the form $s(\mathbf{X}, \mathbf{c})$ and consists of a *predicate* name s, a sequence of variables $\mathbf{X}$, and a sequence of constants $\mathbf{c}$, where $|\mathbf{X}| + |\mathbf{c}|$ is the *arity* of s. We denote variables by capital letters, and constants and predicates by lower-case words. We assume rules to be safe, that is, all variables appear in the positive body. A rule is *ground* if it contains no variables. A *fact* is a ground rule with an empty body and just one head atom. A predicate is *extensional* in Π if it appears only in rule bodies of Π. We also call an atom extensional if its predicate is extensional.

A non-ground rule can be seen as an abbreviation for all possible instantiations of the variables with domain constants. In ASP, this instantiation is explicitly performed by a *grounder* that transforms a (non-ground) program into an equivalent set of ground rules. Grounders implement many advanced techniques to generate ground programs that are as small as possible. Since these techniques are sometimes quite involved and hinder theoretical investigation, for the purposes of this paper we define an idealized grounder as follows:

Definition 2. Let Π be a non-ground ASP program, let Π^+ denote the positive program obtained from Π by removing all negated atoms and replacing disjunctions with conjunctions (i.e., splitting disjunctive into normal rules), and let M^+ be the unique minimal model of Π^+. For every rule $r \in \Pi$ and substitution s from variables to constants, the *grounding* of Π, denoted $\mathrm{gr}(\Pi)$, contains $s(r)$ iff $s(B^+(r)) \subseteq M^+$. □

Ground ASP programs are intended to be interpreted according to the stable model semantics [Gelfond and Lifschitz, 1988]. Given a non-ground program Π (called *encoding*) together with a set of facts $\mathcal{A}$ (called *instance*) as input, the main reasoning task considered in this paper is to decide whether the corresponding grounding $\mathrm{gr}(\Pi \cup \mathcal{A})$ has a stable model (or *answer set*). For fixed programs Π, this problem is Σ_2^P-complete [Eiter and Gottlob, 1995].

In ASP code listings, we represent the $\leftarrow$ symbol in rules by `:-`. In Section 3, we will make use of advanced ASP constructs like inequality and aggregate functions, as described in the ASP language specification [Calimeri *et al.*, 2015].

Graphs, Tree Decompositions, and Treewidth. We assume all graphs to be undirected, simple, and ordered, which means that there is an arbitrary but fixed total order over the vertices of the graph. For a graph G, $V(G)$ denotes the set of vertices and $E(G)$ the set of edges. A graph H is a *minor* of a graph G if H can be obtained from G by deleting edges or vertices, or by contracting edges. The *Cartesian product* $G \square H$ of graphs G and H has vertices $V(G \square H) = V(G) \times V(H)$ and an edge between vertices $\langle u, u' \rangle$ and $\langle v, v' \rangle$ iff $u = v$ and $(u', v') \in E(H)$, or $u' = v'$ and $(u, v) \in E(G)$. A (square) *grid* of size n is the Cartesian product of two paths of length n. The *line graph* G_L of a graph G has $V(G_L) = E(G)$, and $(e_1, e_2) \in E(G_L)$ iff edges e_1 and e_2 share a vertex in G.

Let G be a graph, T a rooted tree, and χ a labeling function

that maps every node t of T to a subset of $V(G)$ called the *bag* of t. The pair (T, χ) is a *tree decomposition* of G if the following holds: (i) for each $v \in V(G)$, there exists a $t \in T$, such that $v \in \chi(t)$; (ii) for each $\{v, w\} \in E(G)$, there exists a $t \in T$, such that $\{v, w\} \subseteq \chi(t)$; and (iii) for each $r, s, t \in T$, such that s lies on the path from r to t, we have $\chi(r) \cap \chi(t) \subseteq \chi(s)$. The *width* of a tree decomposition is defined as the cardinality of its largest bag minus one. The *treewidth* of a graph G, denoted by $tw(G)$, is the minimum width over all tree decompositions of G. For a minor H of a graph G it holds that $tw(G) \geqslant tw(H)$. Trees have treewidth 1. Grids of size n, the complete graph K_n with n nodes, and the complete bipartite graph $K_{n,n}$ all have treewidth n.

Graph Representations and Treewidth of ASP Programs. The *primal graph* of a ground ASP program Π is a graph whose vertices are the atoms in Π and there is an edge $(\underline{a}, \underline{b})$ if atoms $\underline{a}$ and $\underline{b}$ appear together in a rule in Π. The *incidence graph* of Π is a bipartite graph whose vertices are the atoms and rules in Π and there is an edge between a rule r and an atom $\underline{a}$ if $\underline{a}$ appears in r. The *primal treewidth* or *incidence treewidth* of an ASP program Π is the treewidth of its primal or incidence graph, respectively. By *treewidth* we generally refer to the primal treewidth. For ASP programs of bounded treewidth, the answer set existence problem can be solved in linear time [Gottlob *et al.*, 2010]. The *instance graph* of an instance $\mathcal{A}$ is a graph whose vertices are the constants in $\mathcal{A}$, and which has an edge (a, b) if constants a and b appear together in a fact from $\mathcal{A}$. The *treewidth* or *degree* of an instance $\mathcal{A}$ is the treewidth or degree of its instance graph, respectively.

MSO Transductions. Monadic second-order (MSO) logic is an extension of first-order logic by quantification over sets. We omit a formal definition as all formulas in this work will be first-order. MSO can be used to specify *MSO transductions* [Courcelle and Engelfriet, 2012], which are mappings from graphs to graphs. The idea is that an MSO transduction formalizes how a graph G can be transformed into a graph G' by the following operations: (1) copying G a fixed number of times, (2) filtering vertices that satisfy an MSO-definable property, (3) defining the edges of G' in terms of the edges of G via MSO formulas. This is achieved by specifying MSO formulas $(\delta_i)_{i \in I}, (\theta_{i,j})_{(i,j) \in I \times I}$, where I is an arbitrary finite set, each δ_i has one free variable, each $\theta_{i,j}$ has two free variables, and the signature of the formulas consists of the binary predicates edge and succ. For every graph G, we write $\text{Str}(G)$ to denote the relational structure with domain $V(G)$, that interprets edge by $E(G)$ and succ by the successor relation according to the vertex ordering. We call the graphs for which τ is defined *input graphs*, and the image of τ the *output graphs*. The δ_i formulas specify which vertices exist in $\tau(G)$. For each $v \in V(G)$, there can be up to $|I|$ copies of v in $\tau(G)$: There is a copy v_i in $\tau(G)$ iff $\text{Str}(G) \vDash \delta_i(v)$. Finally, the edges in $\tau(G)$ are specified using the $\theta_{i,j}$ formulas. For each pair of vertices $(v, w) \in V(G) \times V(G)$, there is an edge $(v_i, w_j) \in E(\tau(G))$ iff $\text{Str}(G) \vDash \theta_{i,j}(v, w)$ in addition to $\text{Str}(G) \vDash \delta_i(v)$ and $\text{Str}(G) \vDash \delta_j(w)$.

MSO transductions allow us to prove statements about how a graph transformation affects the treewidth of a graph:

Proposition 3 ([Courcelle and Engelfriet, 2012, Corol-

```
1  assign(X,Y,1) | assign(X,Y,0) :- edge(X,Y).
2  sum(V,S\2) :- vertex(V), S = #sum{ A,W :
       assign(W,V,A); A,W : assign(V,W,A) }.
3  :- sum(V,S), capacity(V,C), S != C.
```

Listing 1: Finding valid assignments in capacitated graphs.

lary 1.53]). *Let τ be a fixed MSO transduction. For every input graph G, the treewidth of $\tau(G)$ depends only on the treewidth of G and the degree of $\tau(G)$.*[1]

3 Impact of Treewidth on ASP Solvers

In this section we will demonstrate, by experimental evaluation, that state-of-the-art ASP solvers have an inherent sensitivity to treewidth in practice, that is, they perform faster on ground programs of small treewidth. To show this claim, a carefully designed experiment is needed in order to actually reveal the influence of the treewidth (and not some other parameter) on the solving time. Ground programs used for testing therefore need to (1) have the same number of answer sets, (2) have a uniform structure, (3) have constant size, and (4) vary in treewidth. To this end, we consider the problem of deciding whether a capacitated graph has a valid assignment:

Definition 4. A *capacitated graph* is a pair (G, c), where G is an undirected graph and c a function mapping each vertex to 0 or 1. An *assignment* is a function mapping each edge to 0 or 1, and we call it *valid* if for each $v \in V(G)$ the sum modulo 2 of the values assigned to incident edges equals $c(v)$. $\square$

This problem was used by Urquhart to construct hard SAT formulas [Urquhart, 1987] using the method of Tseitin [1983]. Listing 1 shows an ASP encoding for this problem. Line 1 assigns either 0 or 1 to each edge. Line 2 calculates for each vertex the sum modulo 2 of the values assigned to incident edges. Finally, Line 3 eliminates all assignments where for some vertex the sum and capacity do not agree.

Input Instances. In order to satisfy condition (1), we will only construct unsatisfiable instances. It is known that a connected capacitated graph has a valid assignment iff the sum of all vertex capacities is even [Urquhart, 1987]. To construct unsatisfiable instances, we thus generate graphs where all vertices except for one have capacity 0.

To satisfy condition (2), we choose Listing 1 as our fixed problem encoding and construct instances in the following way: We generate so-called *grid-tree* instances, which are random binary trees of grids as illustrated in Figure 1, according to two parameters: *treesize* (the number of grids in the tree) and *gridsize* (the size of each grid).

Given a grid-tree instance, we can find other grid-tree instances that lead to the same grounding size by increasing the treesize while decreasing the gridsize (or vice versa). Thus, in order to satisfy condition (3), we first fix a grounding size (in terms of the number of atoms), and then find combinations of treesize and gridsize to achieve this grounding size.

Finally, we establish condition (4). For a graph G consisting only of a disjoint union of grids $G_1, \ldots, G_k$ plus

---

[1]In fact, our MSO transductions preserve bounded cliquewidth; however, bounded cliquewidth and bounded treewidth coincide on graphs of bounded degree [Courcelle and Olariu, 2000].
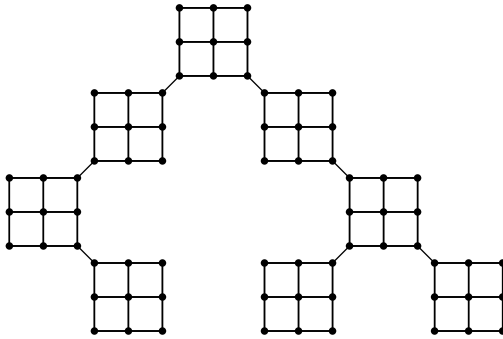
Figure 1: Example for an input instance graph.

```
1 atleast(v7,1) :- 1 { assign(v3,v7,1), assign(v6,v7,1),
         assign(v7,v8,1), assign(v7,v11,1) }.
2 atleast(v7,2) :- 2 { assign(v3,v7,1), assign(v6,v7,1),
         assign(v7,v8,1), assign(v7,v11,1) }.
3 atleast(v7,3) :- 3 { assign(v3,v7,1), assign(v6,v7,1),
         assign(v7,v8,1), assign(v7,v11,1) }.
4 atleast(v7,4) :- 4 { assign(v3,v7,1), assign(v6,v7,1),
         assign(v7,v8,1), assign(v7,v11,1) }.
5 sum(v7,0) :- not atleast(v7,1).
6 sum(v7,0) :- atleast(v7,2), not atleast(v7,3).
7 sum(v7,0) :- atleast(v7,4).
8 sum(v7,1) :- atleast(v7,1), not atleast(v7,2).
9 sum(v7,1) :- atleast(v7,3), not atleast(v7,4).
```

Listing 2: Grounding of Line 2 in Listing 1 for vertex v7.

some edges that do not create any new cycles, it holds that $tw(G) \leq \max\{tw(G_1), \ldots, tw(G_k), 1\}$, since we can easily obtain an appropriate tree decomposition of G from those of $G_1, \ldots, G_k$. Hence the treewidth of a grid-tree instance $\mathcal{A}$ only depends on the gridsize. We now need to show that the treewidth of $\mathcal{A}$ determines the treewidth of the grounding.

Proposition 5. *Given program Π from Listing 1 and a grid-tree instance $\mathcal{A}$, the primal treewidth and incidence treewidth of $\mathrm{gr}(\Pi \cup \mathcal{A})$ are both linear in the treewidth of $\mathcal{A}$.*

Assume, for simplicity, that $\mathcal{A}$ is just a single grid of size 4. We call the nodes in the first row of the grid v1, ..., v4, the nodes in the second row v5, ..., v8, and so on. In order to show the above proposition, first note that Line 1 and Line 3 in Listing 1 cannot cause any cycles in the primal graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$. Now consider Listing 2, which shows the grounding for Line 2 of Listing 1 for vertex v7 of $\mathcal{A}$. Figure 2b shows the primal graph of this partial grounding (where M, N represents the atom assign(vM,vN,1)). Clearly, the rule bodies of the first four lines in Listing 2 cause the central clique between vertices 3, 7; 7, 8; 7, 11; and 6, 7 to appear. Note that this forms a clique precisely between the incident edges of vertex v7. Now, take only the rule bodies of Lines 1 to 4 in Listing 2, and the same also from the analogous groundings of every other vertex in $\mathcal{A}$. The corresponding partial primal graph is shown in Figure 2a. In fact, this is precisely the line graph of the grid in $\mathcal{A}$. The full primal graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$ can now be obtained by replacing every clique in Figure 2a (which represents some vertex v from $\mathcal{A}$) with a gadget analogous to the one in Figure 2b, but for vertex v. Note that this gadget has constant size (and, therefore, treewidth). We thus have that the treewidth of $\mathrm{gr}(\Pi \cup \mathcal{A})$ is asymptotically
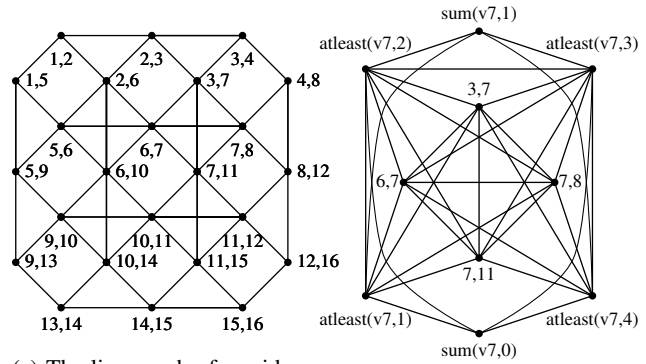
(a) The line graph of a grid of size 4.

(b) Primal graph of Listing 2.
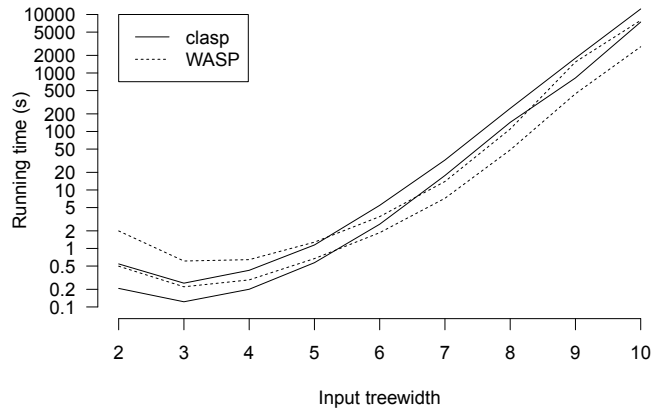
Figure 2: Structure of the primal graph of Listing 2.

Figure 3: Clasp and WASP running time.

upper-bounded by the treewidth of the line graph of $\mathcal{A}$. It is not difficult to extend this argument to the case where $\mathcal{A}$ is a grid-tree instead of a single grid.

To complete our line of argument, note that it is known that the treewidth of the line graph is linear in the treewidth of the original graph if the maximum degree of the latter is bounded by a constant, as is the case for grids [Călinescu *et al.*, 2003]. Since the incidence treewidth is upper-bounded by the primal treewidth [Szeider, 2003], this establishes condition (4).

Benchmark Setting. In our tests we used two batches of instances constructed as presented before, each batch for a different grounding size. For the first batch the number of atoms in the grounding is approximately 10350 and for the second 16700, $\pm$ 100. In each batch there were 20 instances for each treewidth between 2 and 10.[2] Grounding was done using the grounder *gringo* 4.5.4 [Gebser *et al.*, 2011]. We then measured the running time of the ASP solvers *clasp* 3.1.4 [Gebser *et al.*, 2011] and *WASP* 2.0 [Alviano *et al.*, 2015].

Results and Discussion. Figure 3 shows how the average solving time of *clasp* and *WASP* changes with the treewidth for our two batches of instances. Recall that the treewidth of the corresponding ground program is linear in the input

---

[2]Full archive: http://dbai.tuwien.ac.at/proj/decodyn/ijcai17-benchmarks.zip

treewidth by Proposition 5. Since the running time in Figure 3 is denoted on a logarithmic scale, we can thus see that the running time increases exponentially with the treewidth while the grounding size remains the same. Moreover, the running time for small programs with high treewidth can be substantially longer than for large programs with small treewidth.

We thus conclude that the treewidth of the grounding has a major impact on the running time of current ASP solvers and that, for good solving performance, it is important to keep the treewidth of ground programs as low as possible.

4 A Treewidth-Preserving Class of Programs

As we showed in the previous section, treewidth has a strong influence on the solving time of ASP solvers. In order to exploit this relationship in practice, whenever possible, an ASP encoding Π should be written in such a way that, for a given instance $\mathcal{A}$, it does not arbitrarily increase the treewidth of $\mathrm{gr}(\Pi \cup \mathcal{A})$ compared to the treewidth of $\mathcal{A}$. To this end, we introduce the class of connection-guarded ASP programs.

Definition 6. Given an ASP program Π, the *extensional join graph* of a rule r in Π is the graph whose vertices are the variables in r and which has an edge between two variables if these variables occur together in a positive extensional body atom in r. We call Π *connection-guarded* if the extensional join graph of each rule in Π is connected. □

The proposition below follows directly from this definition.

Proposition 7. *Let Π be a connection-guarded program, r a rule in Π, and $\mathcal{A}$ an instance. For any two constants a and b in any ground rule $r' \in \mathrm{gr}(\Pi \cup \mathcal{A})$ obtained from r during grounding, the distance between a and b in the instance graph of $\mathcal{A}$ is at most the number of variables in r.*

The intention of connection-guarded programs is to guarantee that the treewidth of the grounding remains bounded, provided that the treewidth and degree of the input instance is also bounded. The following theorem is the main result of this section and states this formally.

Theorem 8. *Let Π be a fixed connection-guarded program, and $\mathcal{A}$ an instance. If $\mathcal{A}$ has bounded treewidth and degree, then both the primal graph and the incidence graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$ also have bounded treewidth and degree.*

Before we prove this, we define several auxiliary notions. Our proof relies on MSO transductions and in this context we treat instances as relational structures. We build a transduction for program Π that transforms the instance graph of an ASP instance $\mathcal{A}$ into the incidence graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$. For this, we use the following auxiliary formulas.

Firstly, for any positive integer i, the following formula expresses that a vertex y is the i-th neighbor of a vertex x. (Recall that we assume ordered graphs. The transitive closure of the succ relation is MSO-definable and we denote it by $<$.)

$$\mathrm{neigh}_i(x,y) \equiv \mathrm{edge}(x,y) \wedge \neg \exists z \big(z < y \wedge \mathrm{edge}(x,z)$$
$$\wedge \bigwedge_{1 \leqslant j < i} \neg \mathrm{neigh}_j(x,z) \big)$$

A *path identifier* $\pi = \langle i_1, \ldots, i_k \rangle$ *of length* k is a k-tuple of positive integers. It is called *bounded by* some integer d, if

each integer in the tuple is bounded by d. We define the formula $\mathrm{path}_\pi(x,y)$ to express that vertex y is reachable from vertex x via a sequence of vertices $v_0, v_1, \ldots, v_k$ such that $v_0 = x$, $v_k = y$, and v_j is the i_j-th neighbor of v_{j-1}, for $1 \leqslant j \leqslant k$. We write ε to denote the empty tuple.

$$\mathrm{path}_\varepsilon(x,y) \equiv x = y$$
$$\mathrm{path}_{\langle i_1, \ldots, i_k \rangle}(x,y) \equiv \exists z \big(\mathrm{path}_{\langle i_1, \ldots, i_{k-1} \rangle}(x,z)$$
$$\wedge \mathrm{neigh}_{i_k}(z,y) \big) \quad \text{for } k \geqslant 1$$

We define the total order $\prec_d$ over path identifiers bounded by d such that $\pi \prec_d \pi'$ if π is lexicographically smaller than π'. The formula $\mathrm{fp}_\pi^d(x,y)$, defined below, represents that π is (according to $\prec_d$) the first path identifier bounded by d that identifies a path to vertex y when starting from vertex x.

$$\mathrm{fp}_\pi^d(x,y) \equiv \mathrm{path}_\pi(x,y) \wedge \bigwedge_{\pi' \prec_d \pi} \neg \mathrm{path}_{\pi'}(x,y)$$

For all nonnegative integers k, ℓ, d and path identifiers $\pi_1, \ldots, \pi_k$ each of length at most ℓ and bounded by d, we define the formula $\mathrm{uid}_{\pi_1,\ldots,\pi_k}^{\ell,d}(x, y_1, \ldots, y_k)$. It represents that x is the smallest vertex connected to all the vertices $y_1, \ldots, y_k$ via paths of length at most ℓ, and that each π_i is the first path identifier that identifies a path to y_i when starting from x. For ease of notation, let $\mathrm{reach}_\ell(x,y)$ be the formula representing that vertex y is reachable from vertex x in at most ℓ steps along the edge predicate.

$$\mathrm{uid}_{\pi_1,\ldots,\pi_k}^{\ell,d}(x, y_1, \ldots, y_k) \equiv$$
$$\bigwedge_{1 \leqslant i \leqslant k} \mathrm{fp}_{\pi_i}^d(x, y_i) \wedge \neg \exists z \big(z < x \wedge \bigwedge_{1 \leqslant i \leqslant k} \mathrm{reach}_\ell(z, y_i) \big)$$

Note that for the empty tuple ε, the formula $\mathrm{uid}_\varepsilon^{\ell,d}(x) \equiv \neg \exists z \, (z < x)$ holds only for the smallest vertex of the graph.

Having the above definitions at our disposal, we are now ready to proceed to the proof of Theorem 8.

Proof of Theorem 8. We will, for the moment, assume that Π is constant-free; we will show later how to handle the general case. Let ℓ be the maximum number of variables in any rule of Π. Furthermore, assume that $\mathcal{A}$ has bounded treewidth and degree, and let d denote the degree of $\mathcal{A}$.

From Π and d, we will construct a fixed MSO transduction $\tau_{\Pi,d}$ that transforms the instance graph of $\mathcal{A}$ into the incidence graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$ with bounded degree. By Proposition 3, this is sufficient to show our claim. To this end, let $\tau_{\Pi,d} = (\Delta_a \uplus \Delta_r, \Theta)$, where Δ_a contains formulas generating the atom vertices of the incidence graph, Δ_r contains formulas generating the rule vertices, and Θ contains formulas generating the edges between them.

The Set Δ_a. For each predicate p of arity k occurring in Π and for each k-tuple of path identifiers $\langle \pi_1, \ldots, \pi_k \rangle$ each of length at most ℓ and bounded by d, let Δ_a contain the formula

$$\delta_{p[\pi_1,\ldots,\pi_k]}(x) \equiv \exists y_1 \cdots \exists y_k \; \mathrm{uid}_{\pi_1,\ldots,\pi_k}^{\ell,d}(x, y_1, \ldots, y_k).$$

The Set Δ_r. For each rule $r \in \Pi$ with k variables and for each k-tuple of path identifiers $\langle \pi_1, \ldots, \pi_k \rangle$ each of length at most ℓ and bounded by d, let Δ_r contain the formula

$$\delta_{r[\pi_1,\ldots,\pi_k]}(x) \equiv \exists y_1 \cdots \exists y_k \; \mathrm{uid}_{\pi_1,\ldots,\pi_k}^{\ell,d}(x, y_1, \ldots, y_k).$$

The Set Θ. Since we need to link atoms and rules, we have to check compatibility between an instantiation of an atom and the instantiation of a rule. To this end, let p be a predicate of arity k, let r be a rule with variables $Y_1, \ldots, Y_n$, and let $A_{p,r}$ be the set of all those tuples $\langle i_1, \ldots, i_k \rangle$ of integers where the atom $p(Y_{i_1}, \ldots, Y_{i_k})$ occurs in r. We define the formula

$$\mathrm{compat}_{p,r}(x_1, \ldots, x_k, y_1, \ldots, y_n) \;\equiv\; \bigvee_{\langle i_1, \ldots, i_k \rangle \in A_{p,r}} \bigwedge_{1 \leqslant j \leqslant k} x_j = y_{i_j}.$$

This auxiliary formula is true if the variables x_i (representing the instantiation of an atom with k-ary predicate p) agree with the variables y_j (representing the instantiation of a rule r with n variables) in such a way that the p-atom would appear in the instantiation of rule r.

We are now ready to give the formulas in Θ. For each predicate p of arity k occurring in Π and each k-tuple of path identifiers $\langle \pi_1, \ldots, \pi_k \rangle$ each of length at most ℓ and bounded by d, as well as for each rule $r \in \Pi$ with n variables and each n-tuple of path identifiers $\langle \rho_1, \ldots, \rho_n \rangle$ each of length at most ℓ and bounded by d, let Θ contain the formula

$$\theta_{p[\pi_1, \ldots, \pi_k], r[\rho_1, \ldots, \rho_n]}(x, y) \;\equiv\; \exists x_1 \cdots \exists x_k \exists y_1 \cdots \exists y_n \big($$
$$\mathrm{uid}^{\ell,d}_{\pi_1, \ldots, \pi_k}(x, x_1, \ldots, x_k) \wedge \mathrm{uid}^{\ell,d}_{\rho_1, \ldots, \rho_n}(y, y_1, \ldots, y_n)$$
$$\wedge\, \mathrm{compat}_{p,r}(x_1, \ldots, x_k, y_1, \ldots, y_n) \big).$$

This completes the construction of the MSO transduction $\tau_{\Pi,d}$. Let G be the instance graph of $\mathcal{A}$ with degree bounded by d. Clearly, since Π, d and ℓ are fixed, so is $\tau_{\Pi,d}$. From the construction of Θ, it is easy to verify that $\tau_{\Pi,d}(G)$ has bounded degree as well. It now remains to show that $\tau_{\Pi,d}(G)$ yields the incidence graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$ as desired. In fact, $\tau_{\Pi,d}(G)$ does not precisely produce the incidence graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$, as it differs from it in two respects. Firstly, the rules in Δ_a produce a vertex for any atom instantiating a predicate, irrespective of whether these atoms actually appear in $\mathrm{gr}(\Pi \cup \mathcal{A})$. However, this is not a problem since vertices that do not appear in the grounding are isolated in $\tau_{\Pi,d}(G)$ and thus do not increase the treewidth or the degree of $\tau_{\Pi,d}(G)$. The second difference is the fact that $\tau_{\Pi,d}(G)$ does not contain vertices for the facts from $\mathcal{A}$. However, the incidence graph of $\mathrm{gr}(\Pi \cup \mathcal{A})$ contains, for each fact $\underline{a} \in \mathcal{A}$, a rule vertex r_a connected only to the vertex for atom $\underline{a}$. Adding these vertices and edges increases the treewidth at most by one. These observations and the fact that G, by assumption, has bounded treewidth and degree, together with Proposition 3, prove our claim for the incidence graph of constant-free programs Π. Given a tree decomposition $\mathcal{T}$ of the incidence graph, by replacing each rule vertex in a node of $\mathcal{T}$ with all adjacent atom vertices in the incidence graph, we obtain a tree decomposition of the primal graph. The fact that, since Π is fixed, the number of atoms in any rule is bounded proves our claim also for the primal graph.

It is tedious, but straightforward, to generalize this proof to programs with constants, and we will only give the general idea here. For each constant occurring in Π, we need to add a constant symbol to our MSO signature. Then we need to adapt the formulas $\mathrm{uid}^{\ell,d}_{\pi_1, \ldots, \pi_k}$, $\delta_{p[\pi_1, \ldots, \pi_k]}$ and $\theta_{p[\pi_1, \ldots, \pi_k], r[\rho_1, \ldots, \rho_n]}$ in such a way that they allow any π_i or ρ_j to also be a constant from Π (instead of just a path identifier), representing the fact that the corresponding position in the atom or rule is already filled by a constant. $\qquad\square$

The proof of Theorem 8 relies on the rather primitive notion of grounding from Definition 2. Incidence graphs obtained from state-of-the-art grounders are generally subgraphs of the output of our transduction. However, since degree and treewidth of a graph can only decrease for a subgraph, Theorem 8 applies also to state-of-the-art grounders.

Thus connection-guarded programs preserve bounded treewidth of an instance $\mathcal{A}$ in the grounding, under the condition that the degree of $\mathcal{A}$ is also bounded. Unfortunately this condition is necessary, as witnessed by the rule `p(X,Z):- e(X,Y), e(Y,Z)`, where `e` is extensional: When given a tree of height 1 (and thus of treewidth 1) with n vertices, the incidence graph of the grounding has linear treewidth, as the complete bipartite graph $K_{n-1, n-1}$ is a minor of it. Also the restrictions in Definition 6 cannot easily be relaxed without destroying bounded treewidth already with very simple programs: If we allow "unconnected" rules like `p(X,Y):- v(X), v(Y)`, then the complete graph K_n is a minor of the incidence graph of the grounding for any n-vertex instance.

It turns out that, despite their restricted syntax, connection-guarded programs are rather expressive. Straightforward encodings for problems like Graph Coloring or Hamiltonian Cycle directly fall into our class (cf. the second encoding in Example 1). Our final result shows that our class preserves the theoretical complexity bounds of full ASP. Since the encoding for 2QBF in [Leone *et al.*, 2006, Section 3.3.5] is clearly connection-guarded, the following theorem can be immediately obtained:

Theorem 9. *For fixed connection-guarded programs, the answer set existence problem is Σ^P_2-complete.*

5 Discussion

In this paper we experimentally showed that modern ASP solvers perform better when the ground input programs have small treewidth, all other things being equal. This is strong evidence that one should not only aim for small groundings when encoding problems in ASP, but also for groundings of small treewidth. We furthermore defined the class of connection-guarded non-ground ASP programs, and we proved that grounding such programs together with input facts whose representation as a graph has small treewidth and degree leads to a ground program whose treewidth is also small. Thus we provided an effective tool for exploiting the relationship between treewidth and solving performance in order to obtain more efficient ASP encodings. Future work includes the investigation of alternative classes of programs that preserve small treewidth.

Acknowledgments

References

[Alviano *et al.*, 2015] Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In *Proc. LPNMR*, pages 40–54, 2015.

[Atserias *et al.*, 2011] Albert Atserias, Johannes K. Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res. (JAIR)*, 40:353–373, 2011.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Calimeri *et al.*, 2015] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-core-2 input language format. https://www.mat.unical.it/aspcomp2013/ASPStandardization, 2015.

[Călinescu *et al.*, 2003] Gruia Călinescu, Cristina G. Fernandes, and Bruce A. Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded treewidth. *J. Algorithms*, 48(2):333–359, 2003.

[Courcelle and Engelfriet, 2012] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[Courcelle and Olariu, 2000] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discr. Appl. Math.*, 101(1-3):77–114, 2000.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[Eiter and Gottlob, 1995] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995.

[Elkabani *et al.*, 2005] Islam Elkabani, Enrico Pontelli, and Tran Cao Son. Smodels[a] - A system for computing answer sets of logic programs with aggregates. In *Proc. LPNMR*, pages 427–431, 2005.

[Fichte and Szeider, 2015] Johannes K. Fichte and Stefan Szeider. Backdoors to tractable answer set programming. *Artif. Intell.*, 220:64–103, 2015.

[Fichte *et al.*, 2017] Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In *Proc. LPNMR*, 2017. To appear.

[Gebser *et al.*, 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The Potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.

[Gebser *et al.*, 2012a] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Morgan & Claypool, 2012.

[Gebser *et al.*, 2012b] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proc. ICLP*, pages 1070–1080, 1988.

[Gottlob *et al.*, 2010] Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.

[Jakl *et al.*, 2009] Michael Jakl, Reinhard Pichler, and Stefan Woltran. Answer-set programming with bounded treewidth. In *Proc. IJCAI*, pages 816–822, 2009.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[Marek and Truszczyński, 1999] Victor W. Marek and Mirosław Truszczyński. Stable models – an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*, pages 375–398. Springer, 1999.

[Pichler *et al.*, 2014] Reinhard Pichler, Stefan Rümmele, Stefan Szeider, and Stefan Woltran. Tractable answer-set programming with weight constraints: bounded treewidth is not enough. *TPLP*, 14(2):141–164, 2014.

[Selman *et al.*, 1996] Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artif. Intell.*, 81(1-2):17–29, 1996.

[Szeider, 2003] Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In *Proc. SAT*, pages 188–202, 2003.

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

[Tseitin, 1983] Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning*, pages 466–483. Springer, 1983.

[Urquhart, 1987] Alasdair Urquhart. Hard examples for resolution. *JACM*, 34(1):209–219, 1987.

[Wen *et al.*, 2016] Lian Wen, Kewen Wang, Yi-Dong Shen, and Fangzhen Lin. A model for phase transition of random answer-set programs. *ACM Trans. Comput. Log.*, 17(3):22:1–22:34, 2016.

[Zhao and Lin, 2003] Yuting Zhao and Fangzhen Lin. Answer set programming phase transition: A study on randomly generated programs. In *Proc. ICLP*, pages 239–253, 2003.