# ASP Programs with Groundings of Small Treewidth

Bernhard Bliem(✉)

University of Helsinki, Helsinki, Finland
bernhard.bliem@helsinki.fi

**Abstract.** Recent experiments have shown ASP solvers to run significantly faster on ground programs of small treewidth. If possible, it may therefore be beneficial to write a non-ground ASP encoding such that grounding it together with an input of small treewidth leads to a propositional program of small treewidth. In this work, we prove that a class of non-ground programs called *guarded ASP* guarantees this property. Guarded ASP is a subclass of the recently proposed class of connection-guarded ASP, which is known to admit groundings whose treewidth depends on both the treewidth and the maximum degree of the input. Here we show that this dependency on the maximum degree cannot be dropped. Hence, in contrast to connection-guarded ASP, guarded ASP promises good performance even if the input has large maximum degree.

## 1 Introduction

Answer Set Programming (ASP) is a popular formalism for solving computationally hard combinatorial problems with applications in many domains [6,13,17,18]. The workflow for using ASP is generally to first encode the problem at hand in the language of non-ground ASP (i.e., as an ASP program containing variables). Instances of that problem can then be represented as sets of ground (i.e., variable-free) *facts*. To solve an instance, we give our problem encoding together with the input facts to a *grounder*, which produces an equivalent ground program, and we then call an ASP *solver* to compute the solutions.

The solving step is the main workhorse of this approach, so improving solver efficiency is of great interest. One possibility of achieving this is to consider *treewidth* [21], which is a parameter that intuitively measures the cyclicity of a graph: the smaller the treewidth, the closer the graph resembles a tree. By representing ground ASP programs as graphs, we can also use treewidth in the context of ASP. It has turned out that the performance of modern ASP solvers is heavily influenced by the treewidth of the given ground input program. Indeed, an empirical evaluation [2] revealed that the solving time increases drastically when the treewidth of the input increases but the size and the manner of construction of the programs remain the same.

We typically do not encode our problems in ground ASP directly, however, but use non-ground programs. There are usually different ways to encode a problem in non-ground ASP. To present one example given in [2], suppose we want to find all vertices that are reachable from a given starting vertex in a given graph. One way to solve this is by defining the transitive closure of the edge relation:

```
trans(X,Y) ← edge(X,Y).
trans(X,Z) ← trans(X,Y), edge(Y,Z).
  reach(X) ← start(X).
  reach(Y) ← start(X), trans(X,Y).
```

Alternatively, we can avoid defining the transitive closure:

```
reach(X) ← start(X).
reach(Y) ← reach(X), edge(X,Y).
```

When these programs are grounded together with the input, they behave quite differently not only in terms of the size of the grounding but also in terms of the treewidth of the grounding. Indeed, as we will show in this paper, the second program has a property that guarantees that we can find a grounding of small treewidth whenever the input has small treewidth. Programs without this property, like the one above relying on the transitive closure, do not allow for this in general. Hence, the way a problem is encoded can influence the treewidth of the ground program considerably, and as the experiments in [2] have shown, this may also have a massive impact on the solving performance. Even though both programs above solve the same problem, we can thus expect the second one to have much better performance in practice.

Since the input for the solver is obtained by grounding, the way we encode our problem in ASP may lead to groundings of huge treewidth even if our instances actually have small treewidth. Unfortunately, it is not obvious how to write a non-ground ASP encoding in order to achieve a low-treewidth grounding and the benefits that come with it. Some ASP modeling techniques may be safe in the sense that they keep the treewidth small, while others may excessively increase it. We can usually model a problem in different ways, but it is not clear which one should be preferred in terms of treewidth.

In an attempt to remedy this, the authors of [2] show that under certain conditions we can find groundings that have small treewidth whenever the input has small treewidth: They present a class of non-ground ASP programs called *connection-guarded*, whose intuition is to restrict the syntax in such a way that only a limited form of transitivity can be expressed. As shown in [2], for any fixed connection-guarded program there are groundings whose treewidth only depends on the treewidth and the maximum degree of the input facts. So for any fixed connection-guarded program, as long as the maximum degree of the input is bounded by a constant, we can find a grounding that has bounded treewidth whenever the input has bounded treewidth. It is not clear whether this also works for inputs of unbounded maximum degree. After all, it is conceivable that this can be achieved by means of clever grounding techniques. Sadly, no such techniques are known.

In the current work, we show that there is most likely no hope for that: We prove that for some connection-guarded programs there can be no procedure that produces groundings whose treewidth depends only on the treewidth of the input (unless $\mathsf{P} = \mathsf{NP}$ or grounding is allowed to take exponential time).

While the class of connection-guarded programs thus does not achieve the goal of preserving bounded treewidth by grounding, we also present a class that does: We prove that a restriction of connection-guarded programs called *guarded programs* allows us to find groundings that preserve bounded treewidth, and we show that we can still express some problems at the second level of the polynomial hierarchy in that class. We also give indications for when a problem cannot be expressed in guarded ASP.

This paper is structured as follows: First we present some preliminary notions on ASP and treewidth in Sect. 2. Next we discuss grounding and recapitulate a formal definition of this process from [2] in Sect. 3. The first part of our main results is presented in Sect. 4, where we show that connection-guarded ASP does not preserve bounded treewidth. The second part of our contributions follows in Sect. 5, where we prove that guarded ASP preserves bounded treewidth and also provide some complexity results. We discuss the significance and consequences of our results in Sect. 6. Finally we conclude in Sect. 7 and hint at possible directions for future research.

## 2 Preliminaries

### 2.1 Answer Set Programming

We briefly review syntax and semantics of ASP. A *program* in ASP is a set of *rules*, which have the following form:

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, \texttt{ not } b_{k+1}, \ldots, \texttt{not } b_m.$$

The *head* of a rule $r$ is the set denoted by $H(r) = \{a_1, \ldots, a_n\}$, the *positive body* of $r$ is the set $B^+(r) = \{b_1, \ldots, b_k\}$, and the *negative body* of $r$ is the set $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. The *body* of $r$ is now defined as $B(r) = B^+(r) \cup B^-(r)$.

If the head of a rule is empty, then we call the rule a *constraint.* If the body of a rule is empty, then we we may omit the $\leftarrow$ symbol. If the body of a rule is empty and the head consists of a single atom, then we call the rule a *fact.* A program $\Pi$ is called *positive* if the negative body of each rule in $\Pi$ is empty.

All elements of the heads or the bodies of rules are called *atoms.* A *literal* is an atom $a$ or its negated form $\texttt{not } a$. An atom has the form $p(t_1, \ldots, t_\ell)$, where $p$ is called a *predicate.* The elements $t_1, \ldots, t_\ell$ in an atom are called *terms.* A term is either a *constant* or a *variable.* It is customary to write predicates and constants as (strings starting with) lower-case symbols and variables as (strings starting with) upper-case symbols. We call a program or a part of a program (like atoms, rules, etc.) *ground* if it contains no variables. A predicate is called *extensional* in a program $\Pi$ if it only occurs in rule bodies of $\Pi$. A ground fact is an *input fact* for $\Pi$ if its predicate occurs as an extensional predicate in $\Pi$.[1] We write $\|\Pi\|$ to denote the size of $\Pi$ (in terms of bits required for representing $\Pi$ as opposed to the number of rules).

A rule $r$ is *safe* if every variable that occurs in $r$ occurs in an element of $B^+(r)$. A program is safe if all its rules are safe. We only admit ASP programs that are safe.

We define the semantics of ASP in terms of ground programs. For this, we first show how arbitrary programs can be transformed into ground programs.

For any program $\Pi$, a *ground instance* of a rule $r \in \Pi$ is any rule that can be obtained by replacing the variables in $r$ with constants occurring in $\Pi$. The *ground instantiation* $\mathrm{Ground}(\Pi)$ of a program $\Pi$ is the set of all ground instances of all rules in $\Pi$.

We call every subset $I$ of the atoms occurring in $\mathrm{Ground}(\Pi)$ an *interpretation* of $\Pi$. An interpretation $I$ *satisfies* a rule $r$ in $\mathrm{Ground}(\Pi)$ if it contains an element of $H(r) \cup B^-(r)$ or if $B^+(r)$ contains an element that is not in $I$. We say that $I$ is a *model* of $\Pi$ if it satisfies every rule in $\mathrm{Ground}(\Pi)$. We define $\Pi^I$, called the *reduct* of $\Pi$ w.r.t. $I$, as $\Pi^I = \{H(r) \leftarrow B^+(r) \mid r \in \Pi,\ B^-(r) \cap I = \emptyset\}$. We call $I$ an *answer set* of $\Pi$ if $I$ is a model of $\Pi$ and no proper subset of $I$ is a model of $\Pi$. Two ASP programs are *equivalent* if they have the same answer sets.

Deciding if a ground ASP program has an answer set is complete for $\Sigma_2^{\mathsf{P}}$ [10]. Moreover, for any fixed non-ground ASP program $\Pi$, the problem of deciding whether, given a set $F$ of input facts, $\Pi \cup F$ has an answer set is $\Sigma_2^{\mathsf{P}}$-complete [11].

## 2.2 Treewidth

Treewidth is a parameter that measures the cyclicity of graphs. It can be defined by means of *tree decompositions* [21]. The intuition behind tree decompositions is to obtain a tree $T$ from a (potentially cyclic) graph $G$ by subsuming multiple vertices of $G$ under one node of $T$ and thereby isolating the parts responsible for cyclicity.

**Definition 1.** *A* tree decomposition *of a graph $G$ is a pair $\mathcal{T} = (T, \chi)$ where $T$ is a (rooted) tree and $\chi : \mathrm{V}(T) \to 2^{\mathrm{V}(G)}$ assigns to each node of $T$ a set of vertices of $G$ (called the node's* bag*), such that the following conditions are met:*

1. *For every vertex $v \in \mathrm{V}(G)$, there is a node $t \in \mathrm{V}(T)$ such that $v \in \chi(t)$.*
2. *For every edge $(u, v) \in \mathrm{E}(G)$, there is a node $t \in \mathrm{V}(T)$ such that $\{u, v\} \subseteq \chi(t)$.*
3. *If a vertex is contained in the bags of two nodes $t, t'$, then it is also contained in the bags of all nodes between $t$ and $t'$.*

[1] In the database community, one of the origins of ASP, it is common to call a non-ground ASP program an *intensional database* (IDB) and a set of input facts an *extensional database* (EDB). Readers used to this terminology should note that the term "ASP program" generalizes both concepts. When the distinction between a non-ground program and its input is important, we will make this clear by calling the latter (i.e., the EDB) *input facts*.

```
accept ∨ reject.
attend(alice) ∨ attend(bob) ← accept.
```

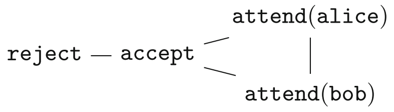

**Fig. 1.** A ground ASP program and its primal graph

*We call* $\max_{t \in V(T)} |\chi(t)| - 1$ *the* width *of* $\mathcal{T}$. *The* treewidth *of a graph is the minimum width over all its tree decompositions.*

It is not hard to see that every tree has treewidth 1 and the complete graph with $n$ vertices (often denoted as $K_n$) has treewidth $n - 1$.

In general, constructing an optimal tree decomposition (i.e., a tree decomposition with minimum width) is intractable [1]. However, the problem is solvable in linear time on graphs of bounded treewidth (specifically in time $w^{\mathcal{O}(w^3)} \cdot n$, where $w$ is the treewidth [3]) and there are also heuristics that offer good performance in practice [4,5,9].

We can easily apply the parameter treewidth to ground ASP programs by defining a suitable representation as a graph.

**Definition 2.** *The* primal graph *of a ground ASP program $\Pi$ is the graph whose vertices are the atoms occurring in $\Pi$ and that has an edge between two atoms if they appear together in a rule in $\Pi$. When we speak of the treewidth of a ground program, we mean the treewidth of its primal graph.*

*Example 3.* Figure 1 depicts a ground ASP program and its primal graph. One possible tree decomposition of the primal graph consists of a chain of two nodes, where one bag contains `accept` and `reject`, and the other bag contains `accept`, `attend(alice)` and `attend(bob)`. Since the largest bag of this tree decomposition has size three, the treewidth of that program is at most two. In fact it is exactly two, since $K_3$ is a subgraph of the primal graph, which means that the treewidth is at least two.

On ground ASP programs, the problem of deciding answer set existence parameterized by the treewidth of the primal graph, is fixed-parameter tractable (FPT; i.e., solvable in time $\mathcal{O}(f(w) \cdot n^c)$, where $f$ is some computable function, $c$ is a constant, and the input has size $n$ and treewidth $w$) [15]. In fact, this problem can even be solved in linear time when the treewidth is bounded by a constant.

As in this work we are interested in the treewidth of groundings in relation to the treewidth of the input of non-ground programs, we also need to define how treewidth can be applied to input facts.

**Definition 4.** *Let $F$ be a set of ground facts. We write $\mathcal{G}(F)$ to denote the graph whose vertices are the constants occurring in $F$ and where there is an edge between two vertices if the respective constants occur together in a fact.*

When we speak of the treewidth or maximum degree of a set $F$ of input facts for a program, we mean the treewidth or maximum degree of $\mathcal{G}(F)$, respectively.

We can now define the property of non-ground programs that is of primary interest in this work.

**Definition 5.** *We say that a (non-ground) ASP program $\Pi$* preserves bounded treewidth *if, for each set $F$ of input facts for $\Pi$ there is a ground program $\Pi'$ such that (1) $\Pi'$ is equivalent to $\Pi \cup F$, (2) we can compute $\Pi'$ in time polynomial in $\|\Pi \cup F\|$, and (3) the treewidth of $\Pi'$ is at most $f(\|\Pi\|, w)$, where $f$ is an arbitrary computable function and $w$ is the treewidth of $\mathcal{G}(F)$.*

We say that a class of ASP programs preserves bounded treewidth if every program in the class does.

## 3 Grounding

The naive ground instantiation Ground($\Pi$) of a program $\Pi$, as defined before, is useful for the definition of the ASP semantics, but it blindly instantiates all variables by all possible constants, which is usually not necessary for obtaining an equivalent ground program. Grounders in practice may omit large parts of Ground($\Pi$) in order to keep the grounding as small as possible while preserving equivalence to Ground($\Pi$). The techniques performed by state-of-the-art grounders are quite sophisticated and differ between systems, so we define a simplified notion of grounding for our study.

For a meaningful investigation of the relationship between the treewidth of input facts and the treewidth of the grounding, we need to assume that the grounder does not simply produce the naive ground instantiation, which instantiates each variable with all constants and thus almost always leads to unbounded treewidth. Instead, we use the following definition of grounding from [2], which formalizes the idea that reasonable grounders will not produce rules whose body is obviously false under every answer set. This simplification is so basic that it can be assumed to be implemented by all reasonable grounders. The intuition is that we omit a rule from the naive ground instantiation whenever its positive body contains an atom that cannot possibly be derived.

**Definition 6.** *Let $\Pi$ be an ASP program, let $\Pi^+$ denote the positive program obtained from $\Pi$ by removing the negative bodies of all rules and replacing disjunctions in the heads with conjunctions (that is, we replace a rule $r$ whose head is $h_1 \vee \ldots \vee h_k$ by rules $r_1, \ldots, r_k$ such that $H(r_i) = \{h_i\}$ and the body of $r_i$ is $B^+(r)$). We say that an atom is* possibly true in $\Pi$ *if it is contained in the unique minimal model of $\Pi^+$. We define the* grounding *of $\Pi$, denoted by* gr($\Pi$), *as the set of all rules $r$ in* Ground($\Pi$) *such that every atom in $B^+(r)$ is possibly true.*

The following example illustrates this.

*Example 7.* Consider the program $\Pi_E$ from Fig. 2a. Following Definition 6, the program $\Pi_E^+$ looks as depicted in Fig. 2b. The unique minimal model of $\Pi_E^+$ consists of $\mathtt{p}(\mathtt{a},\mathtt{b})$, $\mathtt{p}(\mathtt{b},\mathtt{c})$, $\mathtt{q}(\mathtt{b})$, $\mathtt{q}(\mathtt{c})$, $\mathtt{r}(\mathtt{a})$ and $\mathtt{r}(\mathtt{b})$. This allows us to construct the grounding gr($\Pi_E$) as depicted in Fig. 2c. Note that gr($\Pi_E$) does not contain, for instance, the rule $\mathtt{q}(\mathtt{c}) \leftarrow \mathtt{p}(\mathtt{a},\mathtt{c})$, which is present in Ground($\Pi_E$).

## 4 Connection-Guarded ASP with Unbounded Degrees

The class of connection-guarded ASP programs has been introduced in [2] in order to preserve bounded treewidth by grounding, provided that the maximum degree of the input is also bounded. We briefly recapitulate its definition.

**Definition 8.** *Let $\Pi$ be an ASP program. The* join graph *of a rule $r$ in $\Pi$ is the graph whose vertices are the variables in $r$, and there is an edge between two variables if they occur together in a positive extensional body atom of $r$. We call $\Pi$* connection-guarded *if the join graph of each rule in $\Pi$ is connected.*

For any fixed connection-guarded program $\Pi$, given a set $F$ of input facts such that $\mathcal{G}(F)$ has bounded treewidth and bounded maximum degree, the grounding $\mathrm{gr}(\Pi \cup F)$ has bounded treewidth, as shown in [2]. However, it is easy to see that there are connection-guarded programs such that the notion of grounding from Definition 6 leads to ground programs of unbounded treewidth if the degrees are unbounded.

*Example 9.* The program $\Pi$ consisting of the rule $\mathtt{p(X,Z)} \leftarrow \mathtt{edge(X,Y)}, \mathtt{edge(Y,Z)}$ is connection-guarded, but given a set $F$ of facts describing a star (i.e., tree of height 1) with $n$ vertices, the grounding $\mathrm{gr}(\Pi \cup F)$ has unbounded treewidth because, intuitively, it connects all vertices with each other. (More precisely, the complete graph $K_{n-1}$ is a minor of the primal graph of the grounding, which therefore has treewidth at least $n - 2$.)

While this example shows that, for some connection-guarded programs, grounding as described in Definition 6 may destroy bounded treewidth of the input (if the maximum degree is unbounded), it does not rule out that a more sophisticated notion of grounding may actually preserve bounded treewidth on these programs.

In the rest of this section, we show that this cannot be the case (unless $\mathsf{P} = \mathsf{NP}$). We do so by first expressing a problem that is known to be $\mathsf{NP}$-hard on instances of bounded treewidth as a connection-guarded ASP program. Then we prove that $\mathsf{P} = \mathsf{NP}$ holds if this program preserves bounded treewidth. In other words, the existence of a grounder that runs in polynomial time and preserves bounded treewidth of the input for this program implies $\mathsf{P} = \mathsf{NP}$.

```
p(a,b).
p(b,c).
q(Y) ← p(X,Y).
r(X) ← p(X,Y), not q(X).
```

(a) Program $\Pi_E$

```
p(a,b).
p(b,c).
q(Y) ← p(X,Y).
r(X) ← p(X,Y).
```

(b) Program $\Pi_E^+$

```
p(a,b).
p(b,c).
q(b) ← p(a,b).
q(c) ← p(b,c).
r(b) ← p(a,b), not q(a).
r(c) ← p(b,c), not q(b).
```

(c) Grounding $\mathrm{gr}(\Pi_E)$

**Fig. 2.** A program $\Pi_E$, its corresponding positive program $\Pi_E^+$ and grounding

**Theorem 10.** *The problem of deciding whether a fixed connection-guarded program $\Pi$ together with a given set $F$ of input facts has an answer set is* NP*-hard. This even holds if the treewidth of $F$ is at most three.*

*Proof.* We reduce from the following NP-complete problem.

> SUBGRAPH ISOMORPHISM
> Input: Graphs $G$ and $H$
> Question: Is there a subgraph of $G$ that is isomorphic to $H$?

This problem remains NP-hard even if the treewidth of both $G$ and $H$ is at most two [19].

The connection-guarded program in Fig. 3 encodes SUBGRAPH ISOMORPHISM.[2] We use unary predicates `vg` and `vh` to represent the vertices of the input graphs $G$ and $H$, respectively; `eg` and `eh` are binary predicates for the respective edges; the binary predicate `bridge` is used to connect each vertex of $G$ with a new "bridge element", which is in turn connected to each vertex of $H$ also via the `bridge` predicate; and the binary `eq` predicate shall be true for all pairs of identical vertices.

The idea behind the `bridge` predicate is that it allows us to make our encoding connection-guarded in the following way: Whenever we have a rule that contains two variables $X, Y$ that are not connected in the join graph, we can add $\texttt{bridge}(X, \texttt{b})$ and $\texttt{bridge}(\texttt{b}, Y)$ to the positive body and end up with a connection-guarded program.

Now we define a set $F$ of input facts for an instance $\langle G, H\rangle$ of SUBGRAPH ISOMORPHISM according to the intended meaning of our predicates. We use as constants the vertices of $G$ and $H$ as well as a new constant `b`. First we add facts $\{\texttt{vg}(v) \mid v \in \mathrm{V}(G)\}$ and $\{\texttt{vh}(v) \mid v \in \mathrm{V}(H)\}$ to $F$. For the edges, we add facts $\{\texttt{eg}(v, w) \mid (v, w) \in \mathrm{E}(G)\}$ and $\{\texttt{eh}(v, w) \mid (v, w) \in \mathrm{E}(H)\}$. Finally, we add facts $\{\texttt{bridge}(g, \texttt{b}),\ \texttt{bridge}(\texttt{b}, h) \mid g \in \mathrm{V}(G),\ h \in \mathrm{V}(H)\}$ and $\{\texttt{eq}(v, v) \mid v \in \mathrm{V}(G) \cup \mathrm{V}(H)\}$. Note that for every fact $\texttt{eg}(x, y)$ or $\texttt{eh}(x, y)$ in $F$ there is also a fact $\texttt{eg}(y, x)$ or $\texttt{eh}(y, x)$, respectively, since the graphs are undirected. It is easy to verify that this encoding is correct, so $H$ is isomorphic to a subgraph of $G$ if and only if $\Pi \cup F$ has an answer set.

The treewidth of $\mathcal{G}(F)$ is the maximum of the treewidth of $G$ and of $H$ plus one: Given tree decompositions $\mathcal{T}_G$ and $\mathcal{T}_H$ of $G$ and $H$, respectively, we can obtain a tree decomposition of $\mathcal{G}(F)$ by taking the disjoint union of $\mathcal{T}_G$ and $\mathcal{T}_H$, adding the bridge element `b` to every bag and drawing an edge between an arbitrary node from $\mathcal{T}_G$ and an arbitrary node from $\mathcal{T}_H$. $\square$

[2] In practice, we could simplify this encoding substantially by using convenient language constructs provided by ASP systems. For the purpose of this proof, we use our rather restrictive base language. Moreover, note that the positive body of many rules contains atoms whose only purpose is to make the rules connection-guarded. Such redundant atoms could be omitted in practice.

```
% Guess a subgraph S of G using predicates vs/1 and es/2.
        vs(X) ← vg(X), not not_vs(X).
    not_vs(X) ← vg(X), not vs(X).
      es(X,Y) ← eg(X,Y), vs(X), vs(Y), not not_es(X,Y).
  not_es(X,Y) ← eg(X,Y), vs(X), vs(Y), not es(X,Y).
% Guess a relation representing an isomorphism using predicate iso/2.
      iso(G,H) ← vs(G), vh(H), not not_iso(G,H), bridge(G,B), bridge(B,H).
  not_iso(G,H) ← vs(G), vh(H), not iso(G,H), bridge(G,B), bridge(B,H).
% The guessed relation must be a bijection from V(S) to V(H).
          ← iso(G,H1), iso(G,H2), not eq(H1,H2),
            bridge(G,B), bridge(B,H1), bridge(B,H2).
          ← iso(G1,H), iso(G2,H), not eq(G1,G2),
            bridge(G1,B), bridge(G2,B), bridge(B,H).
  used(G) ← iso(G,H), bridge(G,B), bridge(B,H).
  used(H) ← iso(G,H), bridge(G,B), bridge(B,H).
          ← vg(G), vs(G), not used(G).
          ← vh(H), not used(H).
% The guessed relation must be an isomorphism.
  ← iso(G1,H1), iso(G2,H2), es(G1,G2), not eh(H1,H2),
    bridge(G1,B), bridge(G2,B), bridge(B,H1), bridge(B,H2).
  ← iso(G1,H1), iso(G2,H2), eh(H1,H2), not es(G1,G2),
    bridge(G1,B), bridge(G2,B), bridge(B,H1), bridge(B,H2).
```

**Fig. 3.** An encoding of Subgraph Isomorphism in connection-guarded ASP

The fact that solving connection-guarded ASP is NP-hard (in fact $\Sigma_2^{\mathsf{P}}$-complete) when the non-ground part is fixed has already been demonstrated in [2]. The relevance of Theorem 10 is that it shows this problem to be NP-hard *even if the input has bounded treewidth.*

This is particularly interesting because it allows us to prove that, assuming $\mathsf{P} \neq \mathsf{NP}$, there cannot be a grounder that runs in polynomial time and preserves bounded treewidth of the input for every connection-guarded program; if there were, we could solve Subgraph Isomorphism on instances of treewidth at most two, which is still NP-hard, in polynomial time: We reduce the problem to connection-guarded ASP as in the proof of Theorem 10. Grounding would then give us a propositional program of bounded treewidth. As ground ASP can be solved in linear time on instances of bounded treewidth [15], this would allow us to solve the problem in polynomial time.

**Theorem 11.** *If the class of connection-guarded ASP programs preserves bounded treewidth, then* $\mathsf{P} = \mathsf{NP}$.

*Proof.* Suppose connection-guarded ASP preserves bounded treewidth. Then the connection-guarded encoding $\Pi$ of Subgraph Isomorphism from the proof of

Theorem 10 preserves bounded treewidth. In other words, for each set $F$ of facts that encode an instance of Subgraph Isomorphism as described in that proof, there is a ground program $\Pi_F$ such that (1) $\Pi_F$ is equivalent to $\Pi \cup F$, (2) we can compute $\Pi_F$ in time polynomial in $\|\Pi \cup F\|$, and (3) the treewidth of $\Pi_F$ depends only on $\|\Pi\|$ and on the treewidth of $\mathcal{G}(F)$. From the results in [15] it follows that answer set existence for ground programs can be decided in linear time on instances whose primal graph has bounded treewidth. In particular this holds for the ground program $\Pi_F$. Hence we can solve Subgraph Isomorphism in time $\mathcal{O}(f(w) \cdot \|\Pi_F\|)$, where $f$ is some computable function and $w$ is the treewidth of $\Pi_F$. As we can compute $\Pi_F$ in time polynomial in $\|\Pi \cup F\|$, $\|\Pi_F\|$ is polynomial in $\|\Pi \cup F\|$. The program $\Pi$ is the same for every instance, so its size can be considered a constant. Hence $\|\Pi_F\|$ is polynomial in $\|F\|$ and $w$ depends only on the treewidth of $\mathcal{G}(F)$. We have seen in the proof of Theorem 10 that the treewidth of $\mathcal{G}(F)$ depends only on the treewidth of the Subgraph Isomorphism instance. It follows that $w$ is bounded by a constant if the treewidth of the Subgraph Isomorphism instance is bounded by a constant. Therefore Subgraph Isomorphism can be solved in polynomial time on each class of instances of bounded treewidth. As Subgraph Isomorphism is NP-complete on instances whose treewidth is at most two [19], P = NP. □

## 5 Guarded Answer Set Programs

As we have shown in the previous section, connection-guarded ASP does not allow us to preserve bounded treewidth unless the maximum degree of the input is also bounded. In this section, we show that a class of non-ground ASP programs called *guarded ASP* leads to groundings whose treewidth depends only on the treewidth of the input. The notion of guardedness has also appeared, for instance, in the context of the query language Datalog [14].

**Definition 12.** *Let $\Pi$ be an ASP program. We call $\Pi$* guarded *if every rule $r$ in $\Pi$ has an extensional atom $A$ in its positive body such that $A$ contains every variable occurring in $r$.*

Clearly every guarded program is connection-guarded. While guarded ASP is not as expressive, it has the advantage of allowing us to achieve what we could not do with connection-guarded ASP: Guarded ASP preserves bounded treewidth.

**Theorem 13.** *If $\Pi$ is a fixed guarded ASP program containing $c$ constants and $k$ predicates of arity at most $\ell$, and $F$ is a set of input facts for $\Pi$ such that $\mathcal{G}(F)$ has treewidth $w$, then the treewidth of the primal graph of $\mathrm{gr}(\Pi \cup F)$ is at most $k \cdot (w + c + 1)^\ell - 1$.*

*Proof.* Let $\mathcal{T}$ be a tree decomposition of $\mathcal{G}(F)$ having width $w$, and let $C$ denote the constants in $\Pi$. We construct a tree decomposition $\mathcal{T}'$ having width $k \cdot (w + c + 1)^\ell - 1$ of a supergraph of the primal graph of $\mathrm{gr}(\Pi \cup F)$. Since the treewidth of a subgraph is at most the treewidth of the whole graph, the statement follows.

We define the tree in $\mathcal{T}'$ to be isomorphic to the tree in $\mathcal{T}$. Let $N$ be a node in $\mathcal{T}$ and $B$ be its bag. We define the bag $B'$ of the corresponding node $N'$ in $\mathcal{T}'$ to consist of all atoms $p(\boldsymbol{x})$ such that $p$ is a predicate occurring in $\Pi$ and $\boldsymbol{x}$ is a tuple of elements of $B \cup C$. The size of $B'$ is then at most $k \cdot (w + c + 1)^{\ell}$. It remains to show that $\mathcal{T}'$ is indeed a tree decomposition of a supergraph of the primal graph of $\mathrm{gr}(\Pi \cup F)$.

For every atom $p(\boldsymbol{x})$ in a rule $r$ of the grounding, we know from guardedness that there is a ground atom $g(\boldsymbol{y})$ in the positive body of $r$ such that $g$ is extensional and every element of $\boldsymbol{x}$ that is not a constant is also an element of $\boldsymbol{y}$. Since $g$ is extensional, there is a node in $\mathcal{T}$ whose bag contains all elements of $\boldsymbol{y}$. By our construction, the bag of the corresponding node in $\mathcal{T}'$ contains $p(\boldsymbol{x})$.

If two atoms $p(\boldsymbol{x})$ and $q(\boldsymbol{y})$ occur together in a rule $r$ of the grounding, then from guardedness we infer that $r$ also contains an atom $g(\boldsymbol{z})$ in the positive body of $r$ such that $g$ is extensional and every element of $\boldsymbol{x}$ or $\boldsymbol{y}$ that is not a constant is also an element of $\boldsymbol{z}$. As before, it follows that the bag of a node in $\mathcal{T}$ contains all elements of $\boldsymbol{x}$ and $\boldsymbol{y}$ that are not constants, and the bag of the corresponding node in $\mathcal{T}'$ contains both $p(\boldsymbol{x})$ and $q(\boldsymbol{y})$.

If the bags of two nodes $N', M'$ of $\mathcal{T}'$ both contain an atom $p(\boldsymbol{x})$, then the bags of the corresponding nodes $N, M$ in $\mathcal{T}$ contain all elements of $\boldsymbol{x}$ that are not constants. By the definition of tree decompositions, every bag of each node between $N$ and $M$ in $\mathcal{T}$ contains all elements of $\boldsymbol{x}$ that are not constants. Hence, by our construction, the bags of all nodes between $N'$ and $M'$ in $\mathcal{T}'$ contain $p(\boldsymbol{x})$. This proves that $\mathcal{T}'$ is a tree decomposition of a supergraph of the primal graph of $\mathrm{gr}(\Pi \cup F)$, and its width is at most $k \cdot (w + c)^{\ell} - 1$. □

Since the guarded program $\Pi$ and thus $c$, $k$ and $\ell$ are fixed, this shows that the treewidth of the primal graph of $\mathrm{gr}(\Pi \cup F)$ is polynomial in the treewidth of the input $F$.

Combining Theorem 13 with the known fixed-parameter tractability of ground ASP parameterized by treewidth, we immediately get the following result:

**Corollary 14.** *For every fixed guarded ASP program $\Pi$, the problem of deciding for a given set $F$ of input facts whether $\Pi \cup F$ has an answer set is fixed-parameter tractable when parameterized by the treewidth of $\mathcal{G}(F)$.*

This is in contrast to connection-guarded ASP. As we have shown in Theorem 10, for fixed connection-guarded programs the answer set existence problem is most likely not FPT when parameterized by the treewidth of the input facts. Yet this problem is FPT when parameterized by both the treewidth and the maximum degree [2]. To complete the picture of the parameterized complexity of the problem with these two parameters, we analyze the remaining case when just the maximum degree is the parameter. We prove that bounded maximum degree alone is also not sufficient for obtaining fixed-parameter tractability, even if the program is guarded.

```
      t(T) ← verum(T).
      f(F) ← falsum(F).
t(X) ∨f(X) ← exists(X).
t(Y) ∨f(Y) ← forall(Y).
         w ← term(X,Y,Z,Na,Nb,Nc), t(X), t(Y), t(Z), f(Na), f(Nb), f(Nc).
      t(Y) ← w, forall(Y).
      f(Y) ← w, forall(Y).
           ← not w.
```

**Fig. 4.** An encoding of $\mathrm{Q}\textsc{sat}_2$ in guarded ASP

**Theorem 15.** *It is $\Sigma_2^{\mathsf{P}}$-complete to decide for a fixed guarded program $\Pi$ and a given set $F$ of input facts whether $\Pi \cup F$ has an answer set even if the maximum degree of $\mathcal{G}(F)$ is at most 15.*

*Proof.* For membership, we guess an interpretation $I$ and then check by calling a co-$\mathsf{NP}$ oracle whether $I$ is a minimal model of $\mathrm{gr}(\Pi \cup F)^I$. We first show $\Sigma_2^{\mathsf{P}}$-hardness for the case when the maximum degree of $\mathcal{G}(F)$ may be unbounded. Afterwards we show how this construction can be adjusted to obtain degrees of at most 15.

We present a guarded encoding for the well-known $\Sigma_2^{\mathsf{P}}$-complete problem $\mathrm{Q}\textsc{sat}_2$. We are given a formula $\exists x_1 \cdots \exists x_k \forall y_1 \cdots \forall y_\ell\, \varphi$, where $\varphi$ is a formula in 3-DNF (i.e., a disjunction of conjunctive terms, each containing at most three literals), and the question is whether there are truth values for the $x$ variables such that for all truth values for the $y$ variables $\varphi$ is true. We assume that each disjunct in $\varphi$ contains exactly three literals, which can be achieved by using the same literal multiple times in a disjunct.

Consider the ASP program in Fig. 4, which is based on the encoding in Sect. 3.3.5 of [16]. The $\mathrm{Q}\textsc{sat}_2$ formula is represented as a set $F$ of input facts as follows: We will use each variable in $\varphi$ as a constant symbol, and we introduce new constant symbols `t` and `f`. First we put facts `verum`(`t`) and `falsum`(`f`) into $F$. Then, for each existentially or universally quantified variable $x$, we add a fact `exists`($x$) or `forall`($x$), respectively. Finally, for each disjunct $l_1 \wedge l_2 \wedge l_3$ in the formula, we put a fact `term`($p_1, p_2, p_3, q_1, q_2, q_3$) into $F$, where $p_i$ denotes $v_i$ if $l_i$ is a positive atom $v_i$, otherwise $p_i =$ `t`, and $q_i$ denotes $v_i$ if $l_i$ is an atom of the form `not` $v_i$, otherwise $q_i =$ `f`. The fact `term`($p_1, p_2, p_3, q_1, q_2, q_3$) thus represents $p_1 \wedge p_2 \wedge p_3 \wedge \neg q_1 \wedge \neg q_2 \wedge \neg q_3$, which is equivalent to the original disjunct. This program is clearly guarded and indeed encodes the $\mathrm{Q}\textsc{sat}_2$ problem, as can be seen by the arguments in [16].

This shows $\Sigma_2^{\mathsf{P}}$-hardness of answer set existence for fixed guarded programs. We still have to prove that $\Sigma_2^{\mathsf{P}}$-hardness holds even if the maximum degree of $\mathcal{G}(F)$ is at most 15. For this, we first show that we may assume every variable to occur at most three times in $\varphi$ by a construction that has appeared in [20].

Observe that, for each sequence of variables $z_1, \ldots, z_m$, saying that two variables in this sequence have different truth values is equivalent to saying that

(a) some variable $z_i$ is false but $z_{i+1}$ is true, or (b) $z_m$ is false but $z_1$ is true. With this in mind, we obtain a formula $\varphi'$ from $\varphi$ by replacing every occurrence of an (either existentially or universally quantified) variable $z$ by a new variable $z^i$, where $i$ is the number of the respective occurrence in $\varphi$. (That is, the first occurrence of $z$ in $\varphi$ is replaced by $z_1$, the second by $z_2$, and so on.) To establish the connections between the copies of an old variable, we observe that the following statements are equivalent:

1. There are truth values for the $x$ variables such that, for all truth values for the $y$ variables, $\varphi$ is true.
2. There are truth values for the $x$ variables such that, for all truth values for the $y$ variables and for all truth values for the new copies, the following holds: If every old variable $z$ has the same truth value as all of its copies, then $\varphi'$ is true.
3. There are truth values for the $x$ variables such that, for all truth values for the $y$ variables and for all truth values for the new copies, the following holds: The formula $\varphi'$ is true or, for some old variable $z$ with copies $z^1, \ldots, z^m$, two variables in the sequence $z, z^1, \ldots, z^m$ have different truth values.
4. There are truth values for the $x$ variables such that, for all truth values for the $y$ variables and for all truth values for the new copies, the following formula is true, where $\mathrm{Var}(\varphi)$ denotes the variables occurring in $\varphi$:

$$\varphi' \vee \bigvee_{z \in \mathrm{Var}(\varphi) \text{ with } m \text{ copies}} \big((\neg z \wedge z^1) \vee (\neg z^1 \wedge z^2) \vee \cdots \vee (\neg z^{m-1} \wedge z^m) \vee (\neg z^m \wedge z)\big)$$

Thus we obtain an equivalent formula where each variable occurs at most three times.

In contrast to before, where we showed $\Sigma_2^{\mathsf{P}}$-hardness when the maximum degree of $\mathcal{G}(F)$ may be unbounded, we now need to choose slightly different input facts because the domain elements `t` and `f` from the previous construction have unbounded degree. Recall that the old construction puts a fact $\texttt{term}(p_1, p_2, p_3, q_1, q_2, q_3)$ into $F$ for each disjunct in $\varphi$ and that some $p_i$ or $q_j$ may be `t` or `f` in order to represent the equivalent term $p_1 \wedge p_2 \wedge p_3 \wedge \neg q_1 \wedge \neg q_2 \wedge \neg q_2$. The only thing that matters for `t` and `f` is that they are always interpreted as true and false, respectively, which the old construction ensures with the facts `verum(t)` and `falsum(f)`. We can thus just use a certain number of copies of `t` and `f` such that every copy occurs in exactly one fact over the `term` predicate and for each copy $x$ we have the respective fact $\texttt{verum}(x)$ or $\texttt{falsum}(x)$. Clearly this reduction to ASP is still correct. The maximum degree of $\mathcal{G}(F)$ is at most 15 because every vertex has at most five neighbors from each fact over the `term` predicate and every variable occurs in at most three such facts. □

## 6 Discussion

Guardedness is evidently a rather strong restriction, even more so than connection-guardedness. Yet, as we have seen in Theorem 15, the restrictions imposed by guardedness do not alleviate the complexity of deciding answer set existence for fixed non-ground programs compared to the general case. Beside the encoding of $\text{Q}\textsc{sat}_2$ that we have presented, there are also several other relevant problems for which there are straightforward encodings in guarded ASP. But clearly there are also many problems that cannot be expressed in guarded ASP under common complexity-theoretic assumptions. As we have seen in Corollary 14, expressing a problem in guarded ASP amounts to a proof that the problem is FPT when parameterized by treewidth. Hence we most likely cannot find a guarded encoding for any problem that is $\mathsf{W}[1]$-hard for treewidth.

We have argued in Theorem 15 that solving guarded ASP is as hard as ASP in general. In other words, every problem in $\Sigma_2^\mathsf{P}$ can be reduced in polynomial time to guarded ASP, so one might be confused by our claim that we most likely cannot express any $\mathsf{W}[1]$-hard problem in guarded ASP. After all, there are many problems in $\Sigma_2^\mathsf{P}$ that are $\mathsf{W}[1]$-hard when parameterized by treewidth. Note, however, that in general a polynomial-time reduction may increase the treewidth arbitrarily. When talking about the parameterized complexity of problems, we must, however, make sure that our reductions preserve the parameter. So we can indeed find polynomial-time reductions to guarded ASP from some problems that, when parameterized by treewidth, are $\mathsf{W}[1]$-hard, but this requires us to change the problem instances in such a way that the treewidth of the resulting input facts no longer depends only on the treewidth of the original instance.

One may also ask how our result that guarded ASP preserves bounded treewidth relates to grounders in practice. In our investigation of the effect of grounding on the treewidth, we rely on the rather primitive notion of grounding from Definition 6. State-of-the-art grounders, on the other hand, produce groundings whose primal graphs are generally subgraphs of those resulting from our definition of grounding. However, since the treewidth of a subgraph is always at most the treewidth of the whole graph, our result applies also to state-of-the-art grounders.

Moreover, state-of-the-art grounders are capable of solving problems without needing to call an ASP solver if the program has an answer set that is a deterministic consequence of the input, i.e., if no non-deterministic guessing is involved. This is the case, for instance, for Horn programs (that is, ASP programs without negation or disjunction). Our notion of grounding, on the other hand, assumes that the grounder does not propagate deterministic consequences and thus cannot solve such simple problems by itself. This is in fact a reasonable assumption for our purposes: In this work we investigated *syntactic* subclasses of ASP, which means that we are merely interested in the *form* of the non-ground rules. Observe that, for each program that can be solved by the grounder as described before, we can add some rules that force atoms to be guessed. This prevents the grounder from eliminating atoms from rule bodies, and it does not change the form of the original rules. Enforcing the guesses can be done with syntactically very simple (in fact guarded) rules, so *in general* grounders cannot solve guarded programs themselves. (Still, Theorem 13 could of course be slightly extended by allowing for some violations of the guardedness criterion as long as some property of the program guarantees that we can make simplifications that "restore" guardedness.)

Finally, we would like to mention that answer set solving (to be precise, the so-called brave reasoning problem) is still fixed-parameter tractable for guarded (cf. Corollary 14) and even connection-guarded programs when we add weak constraints and aggregates [7] to our language. It is not hard to prove this by translating these constructs into optimization rules and weight rules, respectively, and then invoking the FPT algorithm from [12].

## 7 Conclusion

In this work, we showed that the class of connection-guarded ASP programs does not preserve bounded treewidth (unless $\mathsf{P} = \mathsf{NP}$). That is, for some connection-guarded programs it is impossible to compute a grounding whose treewidth is "small" whenever the input facts have "small" treewidth (unless grounding is allowed to take exponential time or $\mathsf{P} = \mathsf{NP}$). At the same time, we have proven that the more restrictive class of guarded ASP programs achieves the goal of preserving bounded treewidth. It may therefore be a good idea to encode problems in guarded ASP whenever possible, since ASP solvers appear to run much faster on groundings of small treewidth. Unsurprisingly, not all problems can be expressed in guarded ASP. In particular, we proved that problems that are not FPT w.r.t. treewidth cannot be expressed in this class. For several problems it is, however, possible to find straightforward guarded encodings (e.g., several standard graph problems such as graph coloring, vertex cover, dominating set, or some reachability-based problems as shown in the example from Sect. 1). Despite the syntactical restrictions imposed by guardedness, ASP solving in this class does not become easier compared to the general case. As we showed, guarded ASP still allows us to express $\Sigma_2^{\mathsf{P}}$-complete problems.

In the future, it may be interesting to investigate the relationship of guarded and connection-guarded ASP to well-known tools for classifying a problem as FPT for the parameter treewidth. In particular, the famous result by Courcelle [8] states that every problem that is expressible in monadic second-order (MSO) logic is FPT w.r.t. treewidth. Similarly, we have seen in Corollary 14 that also every problem that is expressible in guarded ASP is FPT. Since answer-set solving for guarded programs is $\Sigma_2^{\mathsf{P}}$-complete in general (cf. Theorem 15), whereas MSO model checking is $\mathsf{PSPACE}$-complete, guarded ASP seems to be strictly weaker than MSO. We suspect that this still holds if we add aggregates to guarded ASP. For connection-guarded ASP, however, we conjecture that adding aggregates allows us to encode problems that are not expressible in MSO and its known extensions. We thus expect that connection-guarded ASP can be used as a classification tool for getting FPT results when the parameter is the combination of treewidth and maximum degree.

**Acknowledgments.** This work was supported by the Austrian Science Fund (FWF, project Y698) and by the Academy of Finland (grant 312662). Part of this work was done when the author was employed at TU Wien, Vienna, Austria.

## References

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebr. Discrete Methods **8**(2), 277–284 (1987)
2. Bliem, B., Moldovan, M., Morak, M., Woltran, S.: The impact of treewidth on ASP grounding and solving. In: Sierra, C. (ed.) Proceedings of IJCAI 2017, pp. 852–858. AAAI Press (2017)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. **25**(6), 1305–1317 (1996)
4. Bodlaender, H.L.: Discovering treewidth. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 1–16. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30577-4_1
5. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations I. Upper bounds. Inf. Comput. **208**(3), 259–275 (2010)
6. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
7. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2 input language format, Version: 2.03c (2015). https://www.mat.unical.it/aspcomp2013/ASPStandardization
8. Courcelle, B.: The monadic second-order logic of graphs I: recognizable sets of finite graphs. Inf. Comput. **85**(1), 12–75 (1990)
9. Dermaku, A., Ganzow, T., Gottlob, G., McMahan, B., Musliu, N., Samer, M.: Heuristic methods for hypertree decomposition. In: Gelbukh, A., Morales, E.F. (eds.) MICAI 2008. LNCS (LNAI), vol. 5317, pp. 1–11. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88636-5_1
10. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: propositional case. Ann. Math. Artif. Intell. **15**(3–4), 289–323 (1995)
11. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Trans. Database Syst. **22**(3), 364–418 (1997)
12. Fichte, J.K., Hecher, M., Morak, M., Woltran, S.: Answer set solving with bounded treewidth revisited. In: Balduccini, M., Janhunen, T. (eds.) LPNMR 2017. LNCS (LNAI), vol. 10377, pp. 132–145. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61660-5_13
13. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, Williston (2012)
14. Gottlob, G., Grädel, E., Veith, H.: Datalog LITE: a deductive query language with linear time model checking. ACM Trans. Comput. Log. **3**(1), 42–79 (2002)
15. 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)
16. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3), 499–562 (2006)
17. Lifschitz, V.: What is answer set programming? In: Fox, D., Gomes, C.P. (eds.) Proceedings of AAAI 2008, pp. 1594–1597. AAAI Press (2008)
18. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Apt, K., Marek, V.W., Truszczyński, M., Warren, D.S. (eds.) The Logic Programming Paradigm: A 25-Year Perspective, pp. 375–398. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-642-60085-2_17

19. Matoušek, J., Thomas, R.: On the complexity of finding iso- and other morphisms for partial k-trees. Discrete Math. **108**(1–3), 343–364 (1992)
20. Peters, D.: $\Sigma_2^p$-complete problems on hedonic games, Version: 2. CoRR abs/1509.02333 (2017). http://arxiv.org/abs/1509.02333
21. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. J. Comb. Theory Ser. B **36**(1), 49–64 (1984)