Backdoor Trees for Answer Set Programming*

Johannes K. Fichte and Stefan Szeider
johannes.fichte@tuwien.ac.at sz@ac.tuwien.ac.at

TU Wien, Vienna, Austria

Abstract. We translate the concept of *backdoor trees* from SAT to propositional Answer Set Programming (ASP). By means of backdoor trees we can reduce a reasoning task for a general ASP instance to reasoning tasks on several tractable ASP instances. We demonstrate that the number of tractable ASP instances can be drastically reduced in comparison to a related approach based on *strong backdoors*.

1 Introduction

Answer Set Programming (ASP) is a popular framework for declarative modelling and problem solving [25,33,35]. It has successfully been used to solve a wide variety of problems in artificial intelligence and reasoning, e.g., match making [23], optimization of packaging of Linux distributions [24], reasoning in robots [4], and shift design [1]. In ASP, problems are usually modelled by means of rules and constraints that form a disjunctive logic program. The solutions to the program are the so-called answer sets (or stable models). Solving a problem means to search for answer sets of logic programs. In this paper, we are mainly interested in computational decision problems for propositional disjunctive ASP such as deciding whether a program has an answer set (Consistency), whether a certain atom is contained in at least one (Brave Reasoning) or in all answer sets (Skeptical Reasoning). Further, we consider the problems counting all answer sets (Counting) and enumerating all answer sets (Enum).

Developers of modern ASP solvers such as Clasp [26] or Wasp [3] have demonstrated in several competitions [2,8,10,27,28] that ASP solving can be efficiently used to solve a wide variety of instances. However from the perspective of classical worst case complexity, many decision problems for disjunctive ASP are "harder than NP" and have a higher worst-case complexity than CSP and Sat. More precisely, the problems Consistency, Brave Reasoning, and Skeptical Reasoning are complete for the second level of the Polynomial Hierarchy [13].

In the literature, more fine-grained results on computational complexity of the ASP decision problems have been established. Syntactic properties where the input is restricted to certain fragments have been identified under which the computational complexity drops and where the problems can be solved more

* The first author has been supported by the Austrian Science Fund (FWF), Grant Y698, and is also affiliated with the Institute of Computer Science and Computational Science at University of Potsdam, Germany.

efficiently [29,5,6,39,18]. Parameterized complexity analyses, which take the input size of an instance along with a parameter, indicating the presence of a certain "hidden structure", have been carried out [7,14,15,16,17,32]. The central idea of parameterized complexity is that instances originating in practical applications are often structured in a way that facilitates obtaining a solution relatively fast. An interesting parameter can be gained from backdoors [16]. Backdoors can be used as clever reasoning shortcuts through the search space. For a backdoor one usually fixes a class $\mathcal{C}$ of programs, commonly called target class, where the problem under consideration is computationally easier, e.g., the class of **Horn** programs. Given an input program P, a *strong backdoor* into the target class $\mathcal{C}$ is a preferably small set X of atoms such that for any truth assignment τ to the atoms in X the program under the truth assignment τ (the reduct P_τ, see Definition 1) belongs to $\mathcal{C}$. Then, for program P and a strong backdoor X into $\mathcal{C}$ we have to consider $2^{|X|}$ truth assignments to the atoms in the backdoor X. Exploiting backdoors usually consists of two steps (i) finding a backdoor of the given instance (*backdoor detection*) and (ii) applying the backdoor to the instance, determining a candidate solution, and verifying its minimality (*backdoor evaluation*).

In this paper, we consider backdoor trees, which provide a more fine-grained approach to the evaluation of strong backdoors, where we take partial assignments in the evaluation into account. When using backdoors for a parameterized complexity analysis one only considers the size k of a backdoor as a parameter. Evaluating a given backdoor results in 2^k (total) assignments to the atoms in the backdoor and thus 2^k programs with respect to the possible assignments. However, a partial assignment to fewer than k atoms can already yield a program that belongs to the fixed target class. Therefore, we consider binary decision trees, which make partial assignments to backdoor atoms in a program precise and lead us to the notion of backdoor trees. We investigate under which conditions (i) we need to consider significantly fewer than 2^k assignment reducts and (ii) we can significantly improve parts of the backdoor evaluation (minimality check) if those assignments set only a small number of atoms to true.

Our main contributions are as follows:

1. We define backdoor trees for Answer Set Programming, extend the concept of backdoors from sets to trees (with similar steps detection and evaluation), and establish that the reducts that we obtain from a backdoor tree are sufficient to find all answer sets.
2. We show that backdoor tree evaluation is fixed-parameter tractable when parameterized by a composed parameter, which incorporates considerations of (i) and (ii) from above of a given backdoor tree into **Horn** and other classes.
3. We establish fixed-parameter tractability for backdoor tree detection for backdoor trees into the target class **Horn**.

Related Work. Backdoors were originally introduced by Williams, Gomes, and Selman [40] as a tool for the theoretical analysis of decision heuristics in the area

of SAT and CSP. Nishimura et al. [36] started a systematic investigation of the parameterized complexity of backdoor detection for SAT, which triggered a lot of follow-up work [21]. Samer and Szeider [37] introduced backdoor trees for SAT as a refinement of backdoors and showed that SAT is fixed parameter tractable when parameterized by the number of leaves in a backdoor tree. The problems of detecting backdoor trees into 2CNF and Horn formulas are fixed parameter tractable.

2 Preliminaries

2.1 Answer Set Programming

We consider a universe U of propositional *atoms*. A *literal* is an atom $a \in U$ or its negation $\bar{a}$. A *disjunctive logic program* (or simply a *program*) P is a set of *rules* of the form $a_1 \vee \ldots \vee a_l \leftarrow b_1, \ldots, b_n, \bar{c}_1, \ldots, \bar{c}_m$ where $a_1, \ldots, a_l, b_1, \ldots, b_n, c_1, \ldots, c_m$ are atoms and l, n, m are non-negative integers. We write $H(r) = \{a_1, \ldots, a_l\}$ (the *head* of r), $B^+(r) = \{b_1, \ldots, b_n\}$ (the *positive body* of r), and $B^-(r) = \{c_1, \ldots, c_m\}$ (the *negative body* of r). We denote the sets of atoms occurring in a rule r or in a program P by $\mathrm{at}(r) = H(r) \cup B^+(r) \cup B^-(r)$ and $\mathrm{at}(P) = \bigcup_{r \in P} \mathrm{at}(r)$, respectively. We denote the number of rules of P by $|P| = |\{\, r : r \in P \,\}|$. The *size* $\|P\|$ of a program P is defined as $\sum_{r \in P} |H(r)| + |B^+(r)| + |B^-(r)|$. A rule r is is *normal* if $|H(r)| \leq 1$, r is a *constraint* (integrity rule) if $|H(r)| = 0$, r is *Horn* if it is positive and normal or a constraint. We say that a program has a certain property if all its rules have the property. **Horn** refers to the class of all Horn programs. We denote the class of all normal programs by **Normal**. Let P and P' be programs. We say that P' is a *subprogram* of P (in symbols $P' \subseteq P$) if for each rule $r' \in P'$ there is some rule $r \in P$ with $H(r') \subseteq H(r)$, $B^+(r') \subseteq B^+(r)$, $B^-(r') \subseteq B^-(r)$. Let $\mathcal{C}$ be a class of programs. We call a class $\mathcal{C}$ of programs *hereditary* if for each $P \in \mathcal{C}$ all subprograms of P are in $\mathcal{C}$ as well.

A set M of atoms *satisfies* a rule r if $(H(r) \cup B^-(r)) \cap M \neq \emptyset$ or $B^+(r) \setminus M \neq \emptyset$. M is a *model* of P if it satisfies all rules of P. The *Gelfond-Lifschitz (GL) reduct* of a program P under a set M of atoms is the program P^M obtained from P by first removing all rules r with $B^-(r) \cap M \neq \emptyset$ and then removing all $\bar{z}$ where $z \in B^-(r)$ from the remaining rules r [30]. M is an *answer set* (or *stable model*) of a program P if M is a minimal model of P^M. We denote by $\mathrm{AS}(P)$ the set of all answer sets of P. A class $\mathcal{C}$ of programs is *enumerable* if for each $P \in \mathcal{C}$ we can compute $\mathrm{AS}(P)$ in polynomial time.

In this paper, we consider the following fundamental ASP problems for a given program P: CONSISTENCY asks whether P has an answer set, BRAVE REASONING asks whether a belongs to *some* answer set of P, SKEPTICAL REASONING asks whether a belongs to *all* answer sets of P, COUNTING asks to compute the number of answer sets of P, and ENUM asks to list all answer sets of P. We denote by $\mathcal{AspFull}$ the family of all problems CONSISTENCY, BRAVE REASONING, SKEPTICAL REASONING, COUNTING, and ENUM.

2.2 Parameterized Complexity

We briefly give some background on parameterized complexity. For more detailed information we refer to other sources [11,12,19,31,34]. An instance of a *parameterized problem* L is a pair $(I, k) \in \Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. For an instance $(I, k) \in \Sigma^* \times \mathbb{N}$ we call I the *main part* and k the *parameter*. $\|I\|$ denotes the size of I. L is *fixed-parameter tractable* if there exist a computable function f and a constant c such that we can decide by an algorithm whether $(I, k) \in L$ in time $\mathcal{O}(f(k)\|I\|^c)$. Such an algorithm is called an *fpt-algorithm*. FPT is the class of all fixed-parameter tractable decision problems. The class XP of *non-uniform* polynomial-time tractable problems consists of all parameterized decision problems that can be solved in polynomial time if the parameter is considered constant. That is, $(I, k) \in L$ can be decided in time $\mathcal{O}(\|I\|^{f(k)})$ for some computable function f.

2.3 Backdoors of Answer Set Programs

In the following, we briefly summarize the concept of ASP backdoors [17]. A *(truth) assignment* is a mapping $\tau : X \to \{0, 1\}$ defined for a set $X \subseteq U$ of atoms. For $x \in X$, we define $\tau(\bar{x}) = 1 - \tau(x)$. By 2^X we denote the set of all assignments $\tau : X \to \{0, 1\}$. By $\tau^{-1}(b)$ we denote the preimage $\tau^{-1}(b) := \{\, a : a \in X, \tau(a) = b \,\}$ of the assignment τ for some truth value $b \in \{0, 1\}$.

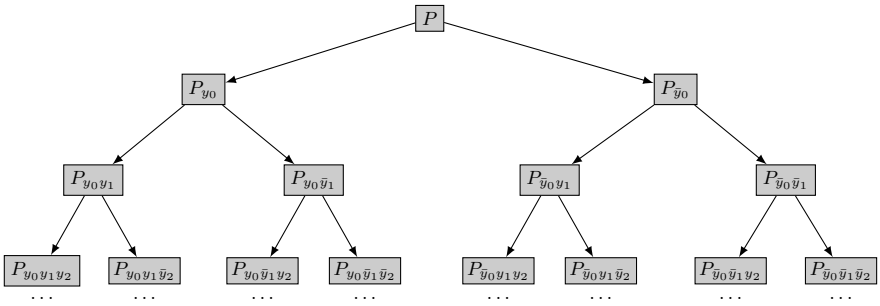
Definition 1 (Strong $\mathcal{C}$-Backdoor [17]) *Let P be a program, X a set of atoms, and $\tau \in 2^X$ an assignment. The* reduct *of P under τ is the logic program P_τ obtained from P by (i) removing all rules r with $H(r) \cap \tau^{-1}(1) \neq \emptyset$ or $H(r) \subseteq X$; (ii) removing all rules r with $B^+(r) \cap \tau^{-1}(0) \neq \emptyset$; (iii) removing all rules r with $B^-(r) \cap \tau^{-1}(1) \neq \emptyset$; (iv) removing from the heads and bodies of the remaining rules all literals $a, \bar{a}$ with $a \in X$. Let $\mathcal{C}$ be a class of programs. A set X of atoms is a* strong $\mathcal{C}$-backdoor *of a program P if $P_\tau \in \mathcal{C}$ for all assignments $\tau \in 2^X$.*

By a *minimal* strong $\mathcal{C}$-backdoor of a program P we mean a strong $\mathcal{C}$-backdoor of P that does not properly contain a smaller strong $\mathcal{C}$-backdoor of P; a *smallest* strong $\mathcal{C}$-backdoor of P is one of smallest cardinality.

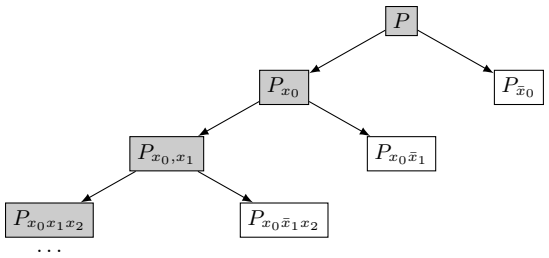
A result by Fichte and Szeider [17] states that all problems in $\mathcal{ASP}\mathit{full}$ are fixed-parameter tractable when parameterized by the size of a given strong $\mathcal{C}$-backdoor for an enumerable target class $\mathcal{C} \subseteq$ **Normal** and that finding a strong backdoor is also fixed-parameter tractable for various target classes, including **Horn**.

3 Backdoor Trees of Answer Set Programs

When we exploit a backdoor X of a program P to find answer sets according to a backdoor-based approach [17], the exponential blowup of the running time depends only on the size of the backdoor X. Thus, we want to find smallest



(a) Constructed from the strong **Horn**-backdoor Y



(b) Constructed from the strong **Horn**-backdoor X

Fig. 1: Illustration of reducts of the program P and the strong **Horn**-backdoors X and Y from Example 1. A gray colored node indicates that the respective program does not belong to **Horn**. A white colored node indicates that the respective program belongs to **Horn**.

backdoors efficiently. Evaluation of backdoors then consists of two steps: (i) computing the answer sets of the program P under the assignment τ for all $\tau \in 2^X$, which produces candidates for answer sets of P (or $\mathrm{AS}(P, X)$[1] for short), and (ii) checking for each $M \in \mathrm{AS}(P, X)$ whether M is a minimal model of P^M. In Step (i) we determine for each $\tau \in 2^X$ the answer sets of the reducts P_τ and then check whether these answer sets give rise to an answer set of P. Consequently, we have to consider all the $2^{|X|}$ assignments in the worst case. However, there are answer set programs where we can find a backdoor X for which we do not need all $2^{|X|}$ assignments, as "shorter" assignments already yield a reduct that belongs to the considered target class. More formally, there is an assignment τ' such that $\tau'^{-1} \subsetneq \tau^{-1}$ for some $\tau \in 2^X$ and the reduct $P_{\tau'}$ already belongs to the target class $\mathcal{C}$. Hence, when we incrementally assign truth values to atoms instead of taking an assignment $\tau \in 2^X$, some atoms in τ can be irrelevant for the question whether the reduct belongs to $\mathcal{C}$.

Interestingly, in some cases it is of advantage to use a backdoor that is not a smallest backdoor into the target class $\mathcal{C}$. We show this in the following example.

[1] Formally, $\mathrm{AS}(P, X) := \{\, M \cup \tau^{-1}(1) : \tau \in 2^{X \cap \mathrm{at}(P)}, M \in \mathrm{AS}(P_\tau) \,\}$.

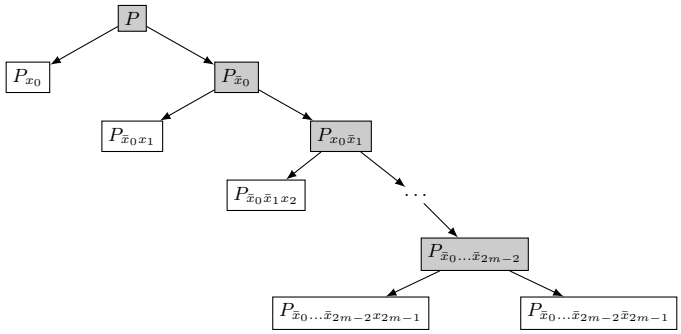


Fig. 2: A **Horn**-backdoor tree $BT = (T, \chi)$ of program R from Example 2.

Example 1. Let m be some positive integer. Consider the following program:

$$\begin{aligned} P := \{\, & y_0 \vee x_0 \leftarrow x_1, \ldots, x_{2m} \,\} \cup \\ & \{\, y_j \vee x_i \leftarrow x_0 \ldots, x_{i-1}, x_{i+1}, \ldots, x_{2m-1};\ x_0 \leftarrow x_1, \ldots, x_{2m-1}, \bar{y}_j; \\ & \quad x_i \leftarrow x_0, x_{i-1}, x_{i+1}, x_{2m-1}, \bar{y}_j : j = (i \bmod m), 0 \leq i < 2m \,\}. \end{aligned}$$

We observe that $Y = \{y_0, \ldots, y_{m-1}\}$ is a smallest strong **Horn**-backdoor. Figure 1(a) visualizes the assignments that we obtain when incrementally constructing the reducts for $\tau \in 2^Y$. Obviously, we need all $2^{|Y|}$ reducts since "removing" any atom from an assignment τ results in $P_\tau \notin$ **Horn**. The set $X = \{x_0, \ldots, x_{2m-1}\}$ is also a strong backdoor into **Horn**. The set X is larger than the set Y, but already for "shorter" assignments τ' than the assignment reducts $\tau \in 2^X$ we obtain that the reduct belongs to **Horn**. For instance, the assignment $\tau' = \{\bar{x}_1\}$ yields the reduct $P_{\bar{x}_1} = \{\, y_1 \leftarrow x_0, x_2, \ldots, x_{2m-1} \,\}$, which belongs to **Horn**. Hence, we obtain only $2m + 1$ reducts, see Figure 1(b). ■

Example 1 shows that incrementally assigning backdoors can yield reducts that belong to the considered target class even though not all atoms of the backdoor are assigned. Then, larger backdoors can yield an exponentially smaller number of such reducts. A main part for backdoor evaluation is to check whether a model is a minimal model ("minimality check"). The task is co-NP-complete in general, but fixed-parameter tractable when parameterized by the size of a smallest backdoor into a subclass of normal programs [17]. For the minimality check we have to consider all backdoor atoms that have been set to true by any assignment. Hence the backdoor Y from Example 1 yields a significantly smaller number of reducts, however for the minimality check we still have to consider all subsets of Y. Conversely, we construct subsequently in Example 2 a program where the number of assignments that we obtain from a smallest strong backdoor can be arbitrarily larger than the maximum number of atoms in a backdoor that have been set to true by any assignment on a much larger number of atoms.

Example 2. Let m be some positive integer. We define the following program:

$$P := \{\, y_0 \vee \bar{x}_0 \leftarrow \bar{x}_1, \ldots, \bar{x}_{2m-1} \,\} \cup$$
$$\{\, y_j \vee x_i \leftarrow \bar{x}_0 \ldots, \bar{x}_{i-1}, \bar{x}_{i+1}, \ldots, \bar{x}_{2m-1};\ x_0 \leftarrow x_1, \ldots, x_{2m-1}, \bar{y}_j;$$
$$x_i \leftarrow \bar{x}_0, \bar{x}_{i-1}, \bar{x}_{i+1}, \bar{x}_{2m-1}, \bar{y}_j : j = (i \bmod m), 0 \leq i < 2m \,\}.$$

$Y = \{y_0, \ldots, y_{m-1}\}$ is a smallest strong **Horn**-backdoor and again Figure 1(a) illustrates how we incrementally construct the reducts until $P_\tau \in$ **Horn**. Same as in Example 1 we have a complete binary tree with 2^m leaves. Further, we easily observe that $X = \{x_0, x_1, \ldots, x_{2m-1}\}$ is a strong **Horn**-backdoor. Figure 2 visualizes the assignments that we obtain when incrementally constructing the reducts $\tau \in 2^X$. There we have only $2m + 1$ reducts where at most one atom is set to true. ■

Before we can make the observations from the previous examples precise, we provide some basic definitions. Let X be a set of atoms, $T = (N, E, r)$ a binary tree with root r, and χ a labeling that maps any node $t \in N$ to a set $\chi(t) \subseteq \{\, a, \bar{a} : a \in X \,\}$. We denote by $X_1(t)$ the positive literals of the labeling $\chi(t)$, i.e., $X_1(t) := \chi(t) \cap X$. The *corresponding assignment* $\tau_{\chi(t)}$ of t is the assignment $\tau_{\chi(t)}$ where $\tau_{\chi(t)}(a) = 1$ if $a \in \chi(t)$ and $\tau_{\chi(t)}(a) = 0$ if $\bar{a} \in \chi(t)$. The pair $BT = (T, \chi)$ is a *binary decision tree* of P if $X \subseteq \mathrm{at}(P)$ and the following conditions hold: (i) for the root r we have $\chi(r) = \emptyset$, (ii) for any two nodes $t, t' \in N$, if t' is a child of t, then either $\chi(t') = \chi(t) \cup \{\bar{a}\}$ or $\chi(t') = \chi(t) \cup \{a\}$ for some atom $a \in X \setminus \tau^{-1}_{\chi(t)}$, and (iii) for any three nodes $t, t_1, t_2 \in N$, if t_1 and t_2 are children of t, then $\chi(t_1) \neq \chi(t_2)$. We denote by $\mathrm{at}(BT)$ the atoms *occurring* in assignments of BT, i.e., $\mathrm{at}(BT) := \bigcup_{t \in N} \tau^{-1}_{\chi(t)}$.

Next, we give a definition for backdoor trees of answer set programs.

Definition 2 *Let P be a program and $BT = (T, \chi)$ be a binary decision tree of P where $T = (N, E, r)$. The pair $BT = (T, \chi)$ is a* $\mathcal{C}$-backdoor tree *of P if $P_\tau \in \mathcal{C}$ for every leaf $t \in N$ and $\tau = \tau_{\chi(t)}$. We denote by* $\#\mathrm{leaves}(BT)$ *the number of leaves of T, i.e., $\#\mathrm{leaves}(BT) := |\{\, t : t \text{ is a leaf of } T \,\}|$. We denote by $\mathrm{gs}(BT)$ the maximum number of atoms that have been set to true by a corresponding assignment of any leaf of T, more specifically, $\mathrm{gs}(BT) := \max\{\, |X_1(t)| : t \text{ is a leaf of } T \,\}$. For reasons explained below, we call $\mathrm{gs}(BT)$ the* Gallo-Scutellà parameter *of BT.*

In other words, a backdoor tree of a program P is a binary decision tree where the nodes of the tree are labeled by assignments $\tau \in 2^X$ on subsets $X \subseteq \mathrm{at}(P)$, the corresponding partial assignment τ of an inner node yields a reduct P_τ that does not belong to the considered target class, and the corresponding assignment τ of a leaf yields a reduct P_τ that belongs to the considered target class.

Relationship to a Parameter by Gallo and Scutellà

Gallo and Scutellà [20] introduced a hierarchy of classes of CNF formulas, with the class of Horn formulas (each clause containing at most one positive literal)

forming its lowest level. They showed that for each level g of the hierarchy, there are polynomial time algorithms fro checking whether a given formula belongs to this level and whether a given formula from this level is satisfiable. The order of the polynomial bounding the running time, however, depends on the level k. Therefore these algorithms do not establish fixed-parameter tractability of these problems, and only render the problems as being in the class XP (see Subsection 2.2).

We consider the parameter in its original context and definition as nested classes of families of sets on a family to generalize Horn formulas. Let S be a family of sets $S_1, \ldots, S_m$, $\mathsf{S}_X = \mathsf{S} \setminus \{ Y \in \mathsf{S} : X \subseteq Y \}$, and $\mathsf{S} - X := \{ S \setminus X : S \in \mathsf{S} \}$ for some set X. Moreover, (i) $\mathsf{S} \in \Sigma_0$ if and only if $|S| \leq 1$ for each $S \in \mathsf{S}$ and (ii) $\mathsf{S} \in \Sigma_k$ if and only if there is some $v \in \bigcup_{1 \leq i \leq m} S_i$ such that $S_{\{v\}} \in \Sigma_{k-1}$ and $\mathsf{S} - \{v\} \in \Sigma_k$. Then, the class Γ_k consists of all propositional formulas F such that $F' \in \Sigma_k$ where F' is obtained from F by removing all negative literals (note that we consider F' as a set of clauses and a clause is a set of literals). Observe that Γ_0 consists of all Horn formulas.

A *backdoor tree of F into Horn formulas* is a binary decision tree where the reduced formula F_τ is Horn for each leaf t and its corresponding assignment τ.

Proposition 3 ($\star$) *A propositional formula F belongs to Γ_k if and only if there is a backdoor tree $BT = (T, \chi)$ into Horn formulas of F such that* $\mathrm{gs}(BT) \leq k$.

4 Backdoor Tree Evaluation

In this section, we establish an analogue to backdoor evaluation for backdoor trees. Again we consider the reducts P_τ together with the atoms that are set to true and extend this notion to the corresponding assignments of the leaves for binary decision trees.

Definition 4 *Let P be a program, $X = \mathrm{at}(P)$, and $BT = (T, \chi)$ a binary decision tree.*

$$\begin{aligned} \mathrm{AS}(P, \tau) :=& \{ M \cup \tau^{-1}(1) : M \in \mathrm{AS}(P_\tau) \} \quad \textit{and} \\ \mathrm{AS}(P, BT) :=& \{ M : \ t \textit{ is a leaf of } T, \tau = \chi(t), M \in \mathrm{AS}(P, \tau) \}. \end{aligned}$$

In other words, the sets in $\mathrm{AS}(P, BT)$ are answer sets of P_τ for assignments τ to $\chi(t) \cap \mathrm{at}(P)$ extended by those atoms which are set to true by τ. In the following lemma we will see that the elements in $\mathrm{AS}(P, BT)$ are all the "answer set candidates" of the original program P. The concepts are similar to ASP backdoors, but slightly more sophisticated.

Lemma 5 ($\star$) *Let P be a program and $BT = (T, \chi)$ a binary decision tree of P. Then,* $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, BT)$.

[1] Due to space limitations, proofs of statements marked with "($\star$)" have been omitted..

Theorem 6 *Given a disjunctive program P, an enumerable class $\mathcal{C} \subseteq$ **Normal**, and a $\mathcal{C}$-backdoor tree $BT = (T, r, \chi)$ of P. Let $g = \mathrm{gs}(BT)$, $\ell = \#\mathrm{leaves}(BT)$, and n be the input size of P. Then, the problems in $\mathcal{A}sp\mathcal{F}ull$ can be solved in time $\mathcal{O}(\ell \cdot 2^g \cdot n^c)$ for some constant c.*

Recall that Example 2 yields programs that have a $\mathcal{C}$-backdoor tree BT with Gallo-Scutellà parameter $\mathrm{gs}(BT) = 1$ and $2m + 1$ leaves whereas a smallest backdoor is of size m. Backdoor evaluation [16, Thm. 3.9] yields a running time $\mathcal{O}(2^{2m}n^c)$ for some constant c and input size n of program P. In contrast, using Theorem 6 we can evaluate the backdoor tree BT in time $\mathcal{O}(2(2m+1) \cdot n^c)$. Consequently, we obtain an exponential speedup for certain programs. In the next section, we will compare backdoor trees with backdoors in more detail.

Before proving Theorem 6, we need to make some observations. In view of Lemma 5 we have to consider the corresponding reducts of the leaves t in the backdoor tree. For each leaf t and its corresponding assignment τ we construct the reduct P_τ and compute the set $\mathrm{AS}(P_\tau)$. Then, we obtain the set $\mathrm{AS}(P)$ by checking for each $M \in \mathrm{AS}(P_\tau)$ whether it gives rise to an answer set of P. The crucial part is again to consider minimality with respect to the Gelfond-Lifschitz reduct. For the leaf t and its corresponding assignment τ we can guarantee minimality with respect to the reduct $(P_\tau)^M$. Setting atoms to true by the assignment τ does apparently not guarantee minimality with respect to P^M (cf. Lemma 5). Hence, we have to check for each atom in $\tau^{-1}(1)$ whether there is a "justification" to set the atom to true.

We establish the following result.

Proposition 7 ($\star$) *Let $\mathcal{C} \subseteq$ **Normal**. Given a program P of input size n, a $\mathcal{C}$-backdoor tree $BT = (T, \chi)$ of P of Gallo-Scutellà parameter $g = \mathrm{gs}(BT)$, a leaf t of T, and a set $M \subseteq \mathrm{AS}(P, \tau_{\chi(t)})$ of atoms, we can check in time $\mathcal{O}(2^g \cdot n)$ whether M is an answer set of P.*

We are now in position to establish Theorem 6.

Proof of of Theorem 6. Let $BT = (T, r, \chi)$ be the given $\mathcal{C}$-backdoor tree, $g = \mathrm{gs}(BT)$, $\ell = \#\mathrm{leaves}(BT)$, $T = (N, E, r)$, and n the input size of P. According to Lemma 5, $\mathrm{AS}(P) \subseteq \mathrm{AS}(P, BT)$. Since $P_\tau \in \mathcal{C}$ and $\mathcal{C}$ is enumerable, we can compute $\mathrm{AS}(P_\tau)$ in polynomial time for each leaf $t \in N$ and $\tau = \tau_{\chi(t)}$, say in time $\mathcal{O}(n^c)$ for some constant c. Hence, $|\mathrm{AS}(P_\tau)| \leq \mathcal{O}(n^c)$ for each leaf $t \in N$ and $\tau = \tau_{\chi(t)}$. By Proposition 7, we can decide whether $M \in AS(P)$ in time $\mathcal{O}(2^g \cdot n^c)$ and $|\mathrm{AS}(P, \tau)| \leq \mathcal{O}(2^g \cdot n^c)$ for each $M \in \mathrm{AS}(P, \tau)$ where $\tau = \tau_{\chi(t)}$ and t is a leaf of T. Since there are at most l many leaves, we can compute $\mathrm{AS}(P, T)$ and check whether for $M \in \mathrm{AS}(P, T)$ also $M \in \mathrm{AS}(P)$ holds in time $\mathcal{O}(\ell \cdot 2^g \cdot n^c)$ and $|\mathrm{AS}(P, T)| \leq \mathcal{O}(\ell \cdot 2^g \cdot n^c)$. Then we can also solve all problems in $\mathcal{A}sp\mathcal{F}ull$ from $\mathrm{AS}(P)$ within polynomial time. Consequently, the problem is fixed-parameter tractable when parameterized by $g + \ell$. □

There are two factors for hardness of ASP problems when parameterized by the Gallo-Scutellà parameter plus the size a backdoor tree (i) atoms that

are set to true which yield potential candidates and are hence important for the minimality check in each leaf; and (ii) leaves in a backdoor tree which yield the reducts we have to consider. Both factors of hardness are "used" in the proof of Theorem 6. Hence, in contrast to SAT backdoor trees we do not simply parameterize the reasoning problems in *AspFull* by $\#\text{leaves}(BT)$ of a given backdoor tree $BT = (T, \chi)$ of P to obtain a more refined view on backdoors. Instead we also consider $\text{gs}(BT)$ which is the maximum number of atoms that are set to true in a leaf of T. This is attributed to the minimality check where we have to consider the number of atoms that are set to true.

5 Relation to Backdoors

In this section, we investigate some connections between backdoors and backdoor trees. We show that our composed parameter based on backdoor trees is more general than the size of a backdoor.

Lemma 8 ($\star$) *Let P be a program and $\mathcal{C}$ be a hereditary class of programs. If BT is a $\mathcal{C}$-backdoor tree of P, then $\text{at}(BT)$ is a strong $\mathcal{C}$-backdoor of P.*

We make the following observations about binary decision trees.

Observation 9 ($\star$) *Let BT be a binary decision tree, $n = |\text{at}(BT)|$, $g = \text{gs}(BT)$, and $\ell = \#\text{leaves}(BT)$. Then, $g \leq n \leq \ell - 1 \leq (1 + n)^g - 1$.*

We establish that every strong backdoor of size k yields a backdoor tree consisting of at least $k + 1$ leaves and at most 2^k leaves.

Lemma 10 ($\star$) *Let P be a program, $\mathcal{C}$ a hereditary class of programs, X a strong $\mathcal{C}$-backdoor of smallest size of P, and $BT = (T, \chi)$ a $\mathcal{C}$-backdoor tree of smallest number of leaves of P. Then, $|X| + 1 \leq \#\text{leaves}(BT) \leq 2^{|X|}$.*

The next observation states that we obtain fewer "answer set candidates" when evaluating ASP backdoor trees than by evaluating ASP backdoors.

Observation 11 ($\star$) *Let P be a program, $BT = (T, \chi)$ a binary decision tree of P, $X := \text{at}(BT)$, and $\text{AS}(P, X) := \{\, M \cup \tau^{-1}(1) : \tau \in 2^{X \cap \text{at}(P)}, M \in \text{AS}(P_\tau) \,\}$. Then, $\text{AS}(P, BT) \subseteq \text{AS}(P, X)$.*

Lemma 12 ($\star$) *Let P be a program, $\mathcal{C}$ a hereditary class of programs, X a strong $\mathcal{C}$-backdoor of smallest size of P, and $BT = (T, \chi)$ a $\mathcal{C}$-backdoor tree of smallest Gallo-Scutellà parameter $\text{gs}(BT)$ of P. Then, $\text{gs}(BT) \leq |X|$.*

Besides having the potential of an exponential speedup (see Example 2), we can trivially construct from a strong $\mathcal{C}$-backdoor X a $\mathcal{C}$-backdoor tree of P with $g = \text{gs}(BT)$ and $\ell = \#\text{leaves}(BT)$ by fixing an arbitrary order on the atoms in X and constructing incrementally all partial assignments τ until $P_\tau \in \mathcal{C}$ or τ assigns all atoms in X. Therefore, we have to consider at most $2^{k+1} - 1$ reducts as we

fixed the order on the backdoor atoms. Using the standard backdoor approach requires to solve the problems in $\mathcal{AspFull}$ a running time $\mathcal{O}(2^{2^{|X|}}n^c)$ [16] for some constant c. By Theorem 6 we can produce solutions in time $\mathcal{O}(2^{g+\log \ell} \cdot n^c)$ for some constant c. Since $g \leq k$ and $\log \ell \leq k$ by Lemmas 12 and 10, $2^{2^{|X|}} \geq 2^{g+\log \ell}$. In that way, backdoor trees can only improve the efficiency compared to the traditional backdoor approach.

6 Backdoor Tree Detection

When we want to exploit backdoor trees to solve a problem instance, we have to detect the backdoor tree first. In this section, we show that detecting **Horn**-backdoor trees is fixed-parameter tractable when parameterized by Gallo-Scutellà parameter and number of leaves. We establish our fixed-parameter tractability results via kernelization, which is in parameterized complexity theory a fundamental method for establishing such results [22]. Intuitively, we can think of a kernel as a "compressed" version of the input, where the size of a kernel is bounded by some computable function of the parameter only and the kernel is produced by a polynomial-time reduction. However, here we need a more restrictive notion of a kernel, loss-free kernels [37], which also occur in the context of subset minimization under the notion of a full kernel [9]. A loss-free kernel contains the union of all minimal solutions and represents in a certain way all solutions.

We first define the following decision problem:

$\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves)

Given:	A program P, an integer $g \geq 0$, and an integer $\ell \geq 0$.
Parameter:	The integer $g + \ell$.
Task:	Decide whether P has a $\mathcal{C}$-backdoor tree BT of Gallo-Scutellà parameter $\text{gs}(BT) \leq g$ and $\#\text{leaves}(BT) \leq \ell$.

By self-reduction (or self-transformation) [38,11,12], we can use a decision algorithm for $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) to actually find the backdoor. Again we only require the target class to be hereditary.

Lemma 13 ($\star$) *Let $\mathcal{C}$ be a hereditary class of programs. If $\mathcal{C}$-Backdoor-Tree Detection(GS,Leaves) is fixed-parameter tractable, then also finding a $\mathcal{C}$-backdoor tree of a given program P of Gallo-Scutellà parameter at most g and at most ℓ leaves is fixed-parameter tractable (when parameterized by $g + \ell$).*

In the following, we consider backdoor tree detection when parameterized by the Gallo-Scutellà parameter g and the number of leaves ℓ of a backdoor tree. Therefore, we consider notions coined by Samer and Szeider [37] in the setting of propositional satisfiability and apply it to Answer Set Programming.

Theorem 14 *The problem* **Horn**-Backdoor-Tree Detection(GS,Leaves) *is fixed-parameter tractable.*

The main ingredient for the proof is that we obtain in polynomial-time an input program a set K of atoms of size at most $k^2 + k$ such that we preserve the union of all minimal strong **Horn**-backdoors of size at most k (Lemma 16). In the previous section, we observed that k is bounded by 2^ℓ. Then, because of the loss-less kernel we can enumerate all $\mathcal{C}$-backdoor trees by brute-force and check for each tree whether the desired parameters are present.

Before we can establish the result, we introduce the notion of a loss-free kernelization for ASP and show how to find such kernels for the problem Strong **Horn**-Backdoor Detection.

Definition 15 *Let $\mathcal{C}$ be a class of programs. A* loss-free kernelization *of the problem* Strong $\mathcal{C}$-Backdoor Detection *is a polynomial-time algorithm that given an instance (I, k), either correctly decides that I does not have a strong $\mathcal{C}$-backdoor of size at most k, or computes a set $K \subseteq \mathrm{at}(P)$ such that the following conditions hold:*

1. *$X \subseteq K$ for every minimal strong $\mathcal{C}$-backdoor X of size at most k and*
2. *there is a computable function f such that $|K| \leq f(k)$.*

Lemma 16 ([37]) *The problem* Strong **Horn**-Backdoor Detection *has a loss-free kernelization with loss-free kernels of size $k^2 + k$.*

We are now in position to establish Theorem 14.

Proof of Theorem 14. Let $\mathcal{C}$ be a class of programs, P a program, $g, \ell > 0$ integers, and $k := 2^\ell$. According to Lemma 16, we compute in polynomial-time for the problem Strong **Horn**-Backdoor Detection a loss-free kernel $K \subseteq \mathrm{at}(P)$ of size at most $k^2 + k$ (if it exists). If P has a $\mathcal{C}$-backdoor tree $BT = (T, \chi)$ of Gallo-Scutellà parameter g and at most ℓ leaves, then $\mathrm{at}(BT)$ is a minimal strong $\mathcal{C}$-backdoor, $g \leq \mathrm{at}(BT)$, and $\#\mathrm{leaves}(BT) \leq \ell - 1$. We claim that $\mathrm{at}(BT) \subseteq K$ for any $\mathcal{C}$-backdoor tree BT with $\mathrm{gs}(BT) \leq g$ and $\ell \leq \#\mathrm{leaves}(BT)$. This follows since $\mathrm{at}(BT)$ contains all minimal strong $\mathcal{C}$-backdoors of size at most k and K is a loss-free kernel. To actually find a suitable $\mathcal{C}$-backdoor tree, we can just try in brute-force all possible $\mathcal{C}$-backdoor trees of P of Gallo-Scutellà parameter at most g and of at most ℓ leaves. Consequently, the theorem follows. □

In view of this result we can drop the assumption in Theorem 6 that the backdoor is provided as input:

Corollary 17 *Let $\mathcal{C} \subseteq$ **Normal** be an enumerable class. The problems in $\mathcal{A}\mathit{sp}\mathcal{F}\mathit{ull}$ are all fixed-parameter tractable when parameterized by $\mathrm{gs}(BT) + \#\mathrm{leaves}(BT)$ of a smallest $\mathrm{gs}(BT) + \#\mathrm{leaves}(BT)$ $\mathcal{C}$-backdoor tree BT.*

Proof. Let P be a program and k an integer. Since there are only linearly many combinations for $k = g + l$, we can use Lemma 13 to find a $\mathcal{C}$-backdoor tree BT of smallest $\mathrm{gs}(BT) + \#\mathrm{leaves}(BT)$ where $\mathrm{gs}(BT) \leq g$ and $\#\mathrm{leaves}(BT) \leq l$ if it exists. The remainder follows from Theorem 6. □

7 Discussion and Future Work

We have introduced backdoor trees to Answer Set Programming. The general concept is similar to the SAT setting but requires additional considerations. The minimality check, which is necessary to verify minimality of potential answer set candidates with respect to the Gelfond-Lifschitz reduct, yields to additional requirements. Therefore, we parameterize the problem of backdoor tree evaluation by a parameter composed of the number of leaves of a backdoor tree and maximum number of atoms that are set to true by a corresponding assignment in a leaf. The former part is crucial to bound the number of potential reducts and hence to bound the number of answer set candidates. The latter part is crucial to bound the number of atoms in any assignment, which we additionally have to consider for the minimality check.

Our parameterization raises the question of whether we can drop one part from the composed parameter. On the one hand, one could parameterize the evaluation problem just by the number of leaves of the backdoor tree, which yields fixed-parameter tractability, but then the evaluation algorithm does not necessarily yield any speedup in the algorithm since we still have to consider the minimality check where a bound on the number of leaves does not pay off when using our minimality check approach. In other words, the evaluation problem is fixed-parameter tractable when parameterized by the number of leaves of backdoor tree. We obtain a parameter that might be significantly smaller, but the running time of the evaluation algorithm can be significantly worse (exponentially). On the other hand, one could parameterize the evaluation problem just by the Gallo-Scutellà parameter (the maximal number of atoms that we have to set to true in any leaf) of the backdoor tree. Since the Gallo-Scutellà parameter of a backdoor tree can be arbitrarily small compared to the number of leaves of a backdoor tree (and hence the size of a smallest backdoor), we obtain an arbitrarily smaller parameter. However, since our upper bound for the number of reducts is $(1 + n)^g$, where n is the number of atoms of the given program and g the Gallo-Scutellà parameter of the backdoor tree, the number of reducts remains non-uniformly bounded. Hence, it remains open whether we obtain fixed-parameter tractability. Moreover, the backdoor tree detection problem when parameterized by the Gallo-Scutellà parameter is only known to be in XP and the question of whether it can be carried out in fixed-parameter tractable time is currently open.

References

1. Abseher, M., Gebser, M., Musliu, N., Schaub, T., Woltran, S.: Shift design with answer set programming. In: Calimeri, F., Ianni, G., Truszczyński, M. (eds.) Proceedings of the 13th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR'15). pp. 32–39. Springer Verlag, Lexington, KY, USA (Sep 2015)
2. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A native ASP solver based on constraint learning. In: Cabalar, P., Son, T. (eds.) Proceedings

of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13), Lecture Notes in Computer Science, vol. 8148, pp. 54–66. Springer Verlag, Corunna, Spain (Sep 2013)
3. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: Calimeri, F., Ianni, G., Truszczyński, M. (eds.) Proceedings of the 13th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR'15). pp. 40–54. Springer Verlag, Lexington, KY, USA (2015)
4. Andres, B., Rajaratnam, D., Sabuncu, O., Schaub, T.: Integrating ASP into ROS for reasoning in robots. In: Calimeri, F., Ianni, G., Truszczyński, M. (eds.) Proceedings of the 13th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR'15). pp. 69–82. Springer Verlag, Lexington, KY, USA (Sep 2015)
5. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. Foundations of deductive databases and logic programming pp. 89–148 (1988)
6. Ben-Eliyahu, R., Dechter, R.: Propositional semantics for disjunctive logic programs. Ann. Math. Artif. Intell. 12(1), 53–87 (1994)
7. Bliem, B., Ordyniak, S., Woltran, S.: Clique-width and directed width measures for answer-set programming. In: Fox, M., Kaminka, G. (eds.) Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI'16). The Hague, Netherlands (Aug 2016), to appear.
8. Calimeri, F., Ianni, G., Ricca, F.: The third open answer set programming competition. Theory Pract. Log. Program. 14, 117–135 (1 2014)
9. Damaschke, P.: Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. Theoretical Computer Science 351(3), 337–350 (2006)
10. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczyński, M.: The second answer set programming competition. In: Erdem, E., Lin, F., Schaub, T. (eds.) Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09), Lecture Notes in Computer Science, vol. 5753, pp. 637–654. Springer Verlag, Potsdam, Germany (Sep 2009)
11. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science, Springer Verlag, New York, NY, USA (1999)
12. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science, Springer Verlag, London, UK (2013)
13. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell. 15(3–4), 289–323 (1995)
14. Fichte, J.K., Hecher, M., Morak, M., Woltran, S.: Answer set solving with bounded treewidth revisited. In: Balduccini, M., Janhunen, T. (eds.) Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17). Lecture Notes in Computer Science, vol. 10377. Springer Verlag, Espoo, Finland (Jul 2017), to appear.
15. Fichte, J.K., Kronegger, M., Woltran, S.: A multiparametric view on answer set programming. In: Bogaerts, B., Harrison, A. (eds.) Informal Proceedings of the 10th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'17) (2017), to appear.
16. Fichte, J.K., Szeider, S.: Backdoors to normality for disjunctive logic programs. ACM Trans. Comput. Log. 17(1) (Oct 2015)
17. Fichte, J.K., Szeider, S.: Backdoors to tractable answer-set programming. Artificial Intelligence 220(0), 64–103 (2015)
18. Fichte, J.K., Truszczyński, M., Woltran, S.: Dual-normal programs – the forgotten class. Theory Pract. Log. Program. (2015)

19. Flum, J., Grohe, M.: Parameterized Complexity Theory, Theoretical Computer Science, vol. XIV. Springer Verlag, Berlin (2006)
20. Gallo, G., Scutellà, M.G.: Polynomially solvable satisfiability problems. Information Processing Letters 29(5), 221–227 (1988)
21. Gaspers, S., Szeider, S.: Backdoors to satisfaction. In: Bodlaender, H., Downey, R., Fomin, F., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond, Lecture Notes in Computer Science, vol. 7370, pp. 287–317. Springer Verlag, Heidelberg, Germany (2012)
22. Gaspers, S., Szeider, S.: Guarantees and limits of preprocessing in constraint satisfaction and reasoning. Artificial Intelligence 216(0), 1–19 (2014)
23. Gebser, M., Glase, T., Sabuncu, O., Schaub, T.: Matchmaking with answer set programming. In: Cabalar, P., Son, T.C. (eds.) Proceedings of 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13). Lecture Notes in Computer Science, vol. 8148, pp. 342–347. Springer Verlag, Corunna, Spain (Sep 2013)
24. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-Criteria Optimization in Answer Set Programming. In: Gallagher, J., Gelfond, M. (eds.) Technical Communications of the 27th International Conference on Logic Programming (ICLP'11). Leibniz International Proceedings in Informatics (LIPIcs), vol. 11, pp. 1–10. Dagstuhl Publishing, Lexington, KY, USA (Jul 2011)
25. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan & Claypool (2012)
26. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications 24(2), 107–124 (2011)
27. Gebser, M., Maratea, M., Ricca, F.: The design of the sixth answer set programming competition. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) Proceedings of the 13th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR'15). pp. 531–544. Springer Verlag, Lexington, KY, USA (Sep 2015)
28. Gebser, M., Maratea, M., Ricca, F.: What's hot in the answer set programming competition. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16). pp. 4327–4329. The AAAI Press, Phoenix, Arizona, USA (Feb 2016), http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12233
29. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88). vol. 2, pp. 1070–1080. MIT Press, Seattle, WA, USA (August 1988)
30. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
31. Gottlob, G., Szeider, S.: Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. The Computer Journal 51(3), 303–325 (2008)
32. Jakl, M., Pichler, R., Woltran, S.: Answer-set programming with bounded treewidth. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09). vol. 2, pp. 816–822. The AAAI Press, Pasadena, CA, USA (Jul 2009)
33. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Apt, K.R., Marek, V.W., Truszczyński, M., Warren, D.S. (eds.) The Logic Programming Paradigm: a 25-Year Perspective, pp. 375–398. Artificial Intelligence, Springer Verlag (1999)

34. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms, Oxford Lecture Series in Mathematics and its Applications, vol. 31. Oxford University Press, New York, NY, USA (2006)
35. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. 25(3), 241–273 (1999)
36. Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to Horn and binary clauses. In: Proceedings of SAT 2004 (Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May, 2004, Vancouver, BC, Canada). pp. 96–103 (2004)
37. Samer, M., Szeider, S.: Backdoor trees. In: Holte, R.C., Howe, A.E. (eds.) Proceedings of 23rd Conference on Artificial Intelligence (AAAI'08). pp. 363–368. The AAAI Press, Chicago, IL, USA (July 2008)
38. Schnorr, C.P.: On self-transformable combinatorial problems. In: König, H., Korte, B., Ritter, K. (eds.) Mathematical Programming at Oberwolfach, Mathematical Programming Studies, vol. 14, pp. 225–243. Springer Verlag (1981)
39. Truszczyński, M.: Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs. Theory Pract. Log. Program. 11, 881–904 (11 2011)
40. Williams, R., Gomes, C., Selman, B.: On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In: Informal Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03). pp. 222–230. Portofino, Italy (May 2003)