Tractable Abstract Argumentation via Backdoor-Treewidth*

Wolfgang Dvořák, Markus Hecher, Matthias König†,
André Schidler, Stefan Szeider, Stefan Woltran

TU Wien, Institute of Logic and Computation
{dvorak, hecher, mkoenig, woltran}@dbai.tuwien.ac.at,
{aschidler, sz}@ac.tuwien.ac.at

Abstract

Argumentation frameworks (AFs) are a core formalism in the field of formal argumentation. As most standard computational tasks regarding AFs are hard for the first or second level of the Polynomial Hierarchy, a variety of algorithmic approaches to achieve manageable runtimes have been considered in the past. Among them, the backdoor-approach and the treewidth-approach turned out to yield fixed-parameter tractable fragments. However, many applications yield high parameter values for these methods, often rendering them infeasible in practice. We introduce the backdoor-treewidth approach for abstract argumentation, combining the best of both worlds with a guaranteed parameter value that does not exceed the minimum of the backdoor- and treewidth-parameter. In particular, we formally define backdoor-treewidth and establish fixed-parameter tractability for standard reasoning tasks of abstract argumentation. Moreover, we provide systems to find and exploit backdoors of small width, and conduct systematic experiments evaluating the new parameter.

1 Introduction

Argumentation is used to resolve conflicts in potentially inconsistent or incomplete knowledge. Argumentation frameworks (AFs), introduced in his influential paper by Dung [10], turned out to be a versatile system for reasoning tasks in an intuitive setting. In AFs we view arguments just as abstract entities, represented by nodes in a directed graph, independent from their internal structure. Conflicts are modelled in form of attacks between these arguments, constituting the edges of said graph representation. The semantics are defined via so called *extensions*—sets of arguments that can be jointly accepted.

*In this version we provide the appendix of [19].

†Corresponding author.

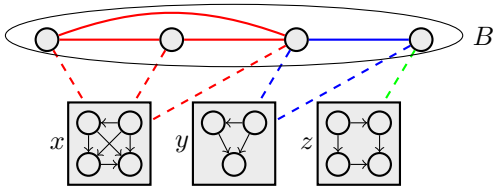


Figure 1: Illustration of the backdoor-treewidth approach

Evaluating AFs w.r.t. these semantics is a central task in argumentation [7]. However, many common reasoning and counting problems regarding extensions turned out to be computationally hard by classical notions [14, 21, 4]. For this reason, various approaches utilizing structural parameters have been introduced [11] and shown to be fruitful for AFs, in particular treewidth [20] and backdoor-distance [15]. These concepts support efficient (i.e., polynomial time) reasoning, provided the respective parameter is small. In this paper, we combine these two approaches to achieve a smaller parameter while still maintaining polynomial runtime in the size of the framework.

To illustrate the usefulness of this concept consider the following example scenario, where two parties argue about a general agreement, where diverging opinions on several subtopics have to be resolved. The discussions about the subtopics might be located in easy fragments of AFs (e.g., symmetric or acyclic frameworks). Then there is a meta-discussion that goes along the lines "if we agree on argument a in subtopic x, we shall have in turn argument b in subtopic y accepted", etc. One can expect that the meta-discussion, while not being in an easy fragment itself, exhibits certain structural features. A high-level illustration of this concept is given in Figure 1, where we have subtopics x, y, z and the meta-discussion B. Here, B indicates the backdoor functioning of this part, i.e. removing B yields an AF composed of easy fragments. In order to employ the backdoor-treewidth approach, we have to use the *torso graph* of B, where any pair of nodes $a, b \in B$ adjacent to same component (x, y, or z) needs to be connected; it is this graph for which we require small treewidth.

Our main contributions can be summarised as follows.

- We define the concept of *backdoor-treewidth* (23) for AFs. Specifically, we present two backdoor-types that allow us to exploit small treewidth, and two backdoor-types that do not.
- Moreover, we argue that given such a backdoor of small treewidth, the reasoning tasks on AFs are *fixed-parameter tractable* (FPT) parameterized by the backdoor's treewidth in complete and stable semantics, extending the known FPT results for the classical notion of treewidth. Fixed-parameter tractability for a problem of input size n and parameter k refers to solvability in time $f(k)n^c$ where f is a (possibly exponential) function of k and c is a constant [25].
- We utilize SAT-solvers both to compute a suitable backdoor set of small treewidth and to solve the instance. The latter is done via a tree decomposition-guided reduction [22] from the argumentation-specific problems to propositional satisfiability. The resulting propositional formula is then evaluated with an adaptation

of the NestHdb system for dynamic programming [26].
- Our experimental results indicate that backdoor-treewidth for argumentation is promising compared to existing techniques based on the measures treewidth and backdoor-size. In particular, it turns out that in practice, backdoor-treewidth can solve many additional instances, where both treewidth and backdoor-size is insufficient.

Some proofs are only given in full length in the appendix.

2 Background

Argumentation. An argumentation framework (AF) due to Dung [10] is a pair $F = (A, R)$ where A is a non-empty and finite set of arguments, and $R \subseteq A \times A$ is the attack relation. We write $S \mapsto_R b$ if there is some $a \in S$ such that $(a, b) \in R$. Likewise, $S \mapsto_R S'$ denotes $S \mapsto_R b$ for some $b \in S'$. Given an AF $F = (A, R)$, a set $S \subseteq A$ is *conflicting* in F if $S \mapsto_R a$ for some $a \in S$. A set $S \subseteq A$ is *conflict-free* in F, if S is not conflicting in F, i.e. if $\{a, b\} \not\subseteq S$ for each $(a, b) \in R$. An argument $a \in A$ is *defended* (in F) by $S \subseteq A$ if for each $B \subseteq A$, $B \mapsto_R a$ implies $S \mapsto_R B$. A set T of arguments is defended (in F) by S if each $a \in T$ is defended by S (in F). Let $S \subseteq A$ be a conflict-free set in F. Then, S is *admissible* in F, denoted by $S \in \mathit{adm}(F)$, if S defends itself in F; S is *stable* in F ($S \in \mathit{stb}(F)$), if S attacks every argument in $A \setminus S$; S is *complete* for F ($S \in \mathit{com}(F)$), if $S \in \mathit{adm}(F)$ and contains every argument it defends; S is *preferred* in F ($S \in \mathit{pref}(F)$), if $S \in \mathit{adm}(F)$ and there is no $T \in \mathit{adm}(F)$ with $T \supset S$. Standard reasoning tasks are *credulous/skeptical acceptance* (is an argument in one/all extensions?).

Backdoors to Tractability. Reasoning on AFs turned out to be NP/coNP-hard for most cases (we assume the reader to be familiar with complexity classes P, NP, coNP, and Π_2^{P}; see the work of Dvořák and Dunne [14] for an introduction to complexity theory in the context of argumentation). Hence, work has been done to identify means of achieving computational speedups [9, 11, 12], which lead to the discovery of *tractable fragments of argumentation*. We consider here acyclicity, even-cycle-freeness, bipartiteness, and symmetry, denoted by *ACYC*, *NOEVEN*, *BIP*, and *SYM*, respectively, and defined in a standard way for directed graphs [14]. As these fragments restrict the possible argumentation structures, to exploit their computational advantages while retaining expressiveness we consider arbitrary (fixed) distances.

We use the *backdoor*-approach by Dvořák et al. [15]: let $F = (A, R)$ be an AF and let $\mathcal{C}$ be a tractable fragment. We call a set $S \subseteq A$ a $\mathcal{C}$-backdoor if $(A', R \cap (A' \times A'))$ for $A' = A \setminus S$ belongs to $\mathcal{C}$. We denote the size of a smallest $\mathcal{C}$-backdoor by $\mathtt{bd}_{\mathcal{C}}(F)$. If it is clear what fragment $\mathcal{C}$ is meant, we shall drop the subscript. Reasoning w.r.t. stable, complete, and preferred semantics is tractable for the fragments *ACYC* and *NOEVEN* if $\mathtt{bd}_{\mathcal{C}}(F)$ is fixed [15].

Treewidth. We recall the notion of *treewidth* [30]. Let $F = (A, R)$ be an AF. Let $U^F = (V, E)$ be the corresponding undirected graph, i.e., $V = A$ and there is an edge between two vertices $a, b \in V$ iff $a \neq b$ and there is an attack (a, b) in R. A *tree*

decomposition (TD) of F is a pair $(\mathcal{T}, \mathcal{X})$, where $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is a tree and $\mathcal{X} = (X_t)_{t \in V_\mathcal{T}}$ is a set of bags (a bag is a subset of A) such that (1) $\bigcup_{t \in V_\mathcal{T}} X_t = V$; (2) for each $v \in V$, the subgraph induced by v in $\mathcal{T}$ is connected; and (3) for each $\{v, w\} \in E$, $\{v, w\} \subseteq X_t$ for some $t \in V_\mathcal{T}$. The width of a TD is $\max\{|X_t| \mid t \in V_\mathcal{T}\} - 1$, the *treewidth* of F, denoted by $\mathtt{tw}(F)$, is the minimum width of all TDs for F. Reasoning on an AF F is fixed-parameter tractable parameterized by its treewidth $\mathtt{tw}(F)$ [20]. For utilizing these results, weaker notions of extensions exist:

Definition 1 (Restricted Extensions)**.** *Let $F = (A, R)$ be an AF and $C, S \subseteq A$ be sets of arguments. We call S a C-*restricted stable set*, if S is conflict-free in F and S attacks all arguments $a \in C$. Then, S is a C-*restricted complete set *if S is conflict-free in F, defends all arguments in $C \cap S$ and contains every argument in C that is defended by S.*

3 Towards Backdoor-Treewidth

In this section, we formally define backdoor-treewidth for abstract argumentation. Intuitively, this notion allows us to "hide" subframeworks (that belong to tractable fragments) and decompose the graph, called *torso graph*, containing "remaining" arguments (i.e., *backdoor arguments*). Ultimately, we utilize a tree decomposition of this torso graph to perform dynamic programming (DP). To ensure correct interaction between these backdoor arguments and the tractable fragments, the whole neighborhood of such a subframework is completely connected in the torso graph. This forces that neighboring arguments of subframeworks appear in a common bag of the TD and enables solving.

Torso Graphs and their Width. We start with the notion of S-components, i.e., components that stay connected (ignoring directions of attacks) after deleting a set S of arguments. Let $F = (A, R)$ be an AF, $S \subseteq A$ be a set of arguments and let $U^F = (V, E)$ be the *corresponding undirected graph*. An *S-component* is a connected component (maximal connected subgraph) of the graph $U^{F'}$, which we obtain from U^F by deleting the vertices S and their incident edges. The torso graph is an aggregated representation of F.

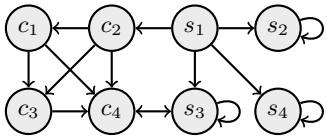
Definition 2 (Torso Graph)**.** *Let $F = (A, R)$ be an AF and $S \subseteq A$ a set of arguments. The S-*torso graph *G_F^S is defined as the (undirected) graph with S as vertices, where two vertices s, t are adjacent iff there is an S-component that s and t are adjacent to in U^F, or an attack (s, t) or (t, s) in R.*

This definition allows us to define backdoor-treewidth in terms of TDs over all possible torso graphs.

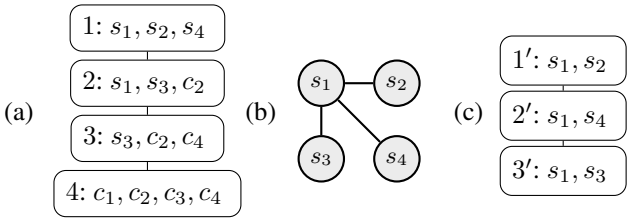
Definition 3 (Backdoor-Treewidth)**.** *The $\mathcal{C}$-*backdoor-treewidth $\mathtt{btw}_\mathcal{C}(F)$ *of an AF F is the minimal width of all tree decompositions of the torso graphs G_F^S for all $\mathcal{C}$-backdoors S of F.*

For the ease of presentation, in this work we assume TDs, where each subset-maximal bag appears (also) in the bag of a leaf of the TD. Indeed, this is not a hard restriction and any TD of the torso graph can be transformed such that this property holds, without increasing its width. The arguments of tractable subframeworks not contained in the torso graph are then considered in leaf node bags containing relevant backdoor arguments (i.e., those adjacent to the S-component). To this end, let $F = (A, R)$ be an AF, $S \subseteq A$ be a backdoor and $(\mathcal{T}, \mathcal{X})$ be a tree decomposition of G_S^F. Then, for a leaf node $t \in \mathcal{T}$, we denote by *components*(t) the *union of all arguments in S-components*, where all adjacent vertices that are in S are also in X_t. As the neighborhood of every S-component s is a clique, there is at least one such leaf node t such that X_t contains the arguments in s.

Example 1. Consider the following AF $F = (A, R)$, consisting of an acyclic 4-clique c_1, c_2, c_3, c_4 connected to a star s_1, s_2, s_3, s_4 with self-attacking leaves.



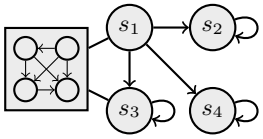
Because of the self-attacks, the minimum *ACYC*-backdoor has size at least 3. In fact, one can verify $B = \{s_2, s_3, s_4\}$ is such an *ACYC*-backdoor. As F contains a 4-clique, the treewidth of F is at least 3. Indeed, the following tree decomposition (a) of width 3 assures $\mathtt{tw}(F) = 3$.



Now consider $B' = \{s_1, s_2, s_3, s_4\}$. Obviously, B' is an *ACYC*-backdoor (but not minimal, as $B' \supset B$). The torso graph w.r.t. B' is exactly the star graph with s_1, s_2, s_3, s_4 as its vertices, i.e., it has backdoor-treewidth 1 (see (b), (c) above). We have *components*$(3') = \{c_1, c_2, c_3, c_4\}$.

Observe that we could add arbitrarily many new arguments to the clique and by doing so increasing the treewidth, while still remaining $\mathtt{btw}_{ACYC}(F) = 1$. Likewise, we can add self-attacking arguments as leaves to the star, increasing the minimum backdoor size of F, while, again, the backdoor-treewidth stays the same. Moreover, we want to highlight that the backdoor B' is not minimal - in fact, the cardinality-minimal backdoor B has a higher width.

We achieve this lower width by hiding easy acyclic subframeworks. In (b), the acyclic B'-component is adjacent only to s_1 and s_3, which assures a small width in the torso graph $G_F^{B'}$ (see illustration below).



Moreover, the backdoor-treewidth $\mathtt{btw}_{\mathcal{C}}(F)$ of an AF $F = (A, R)$ is bounded by $\min(\mathtt{bd}_{\mathcal{C}}(F), \mathtt{tw}(F))$: as A as a whole is a backdoor to any tractable fragment, U^F is the torso G^A_F. Moreover, the tree decomposition of G^B_F (for B being a minimum size backdoor) with only one bag containing all of B has width $|B|-1$. From this it follows:

Proposition 1. *For an AF F and a tractable class $\mathcal{C}$,*
1. $\mathtt{bd}_{\mathcal{C}}(F)$ *and* $\mathtt{tw}(F)$ *can be arbitrarily large even while* $\mathtt{btw}_{\mathcal{C}}(F)$ *remains constant,*
2. $\mathtt{btw}_{\mathcal{C}}(F) < \mathtt{bd}_{\mathcal{C}}(F)$*, and*
3. $\mathtt{btw}_{\mathcal{C}}(F) \leq \mathtt{tw}(F)$.

In the following, we use *chordal* AFs: an AF $F = (A, R)$ is *chordal* if each set $C \subseteq A$ of arguments inducing a directed cycle of F contains a set of arguments C', $|C'| \leq 3$, that induces a directed cycle of F as well. Hence Proposition 1 also applies in the special case of chordal AFs.

4 Finding Backdoors of Minimum Width

Unfortunately, computing backdoor-treewidth exactly is a nontrivial task since it is insufficient to restrict to sets of size at most k. The non-parameterized version of this problem for *ACYC*-backdoors is NP-hard.

Proposition 2. *Deciding whether a given AF F has an ACYC-backdoor of width $\leq k$ is* NP*-complete (if k is part of the input).*

However, it is known that finding the minimum backdoor-treewidth for backdoors in other contexts such as SAT or CSP is fixed-parameter tractable parameterized by backdoor-treewidth [24, 23]. We utilize this result to obtain fixed-parameter tractability results for *SYM* in the general case and for *ACYC* for chordal AFs. The proofs for these theorems are given in the appendix and utilize respective results for CSP [23].

Theorem 1. *Given an AF F and a parameter k, it is fixed-parameter tractable to either return a SYM-backdoor of width $\leq k$ or correctly conclude that the SYM-backdoor-treewidth of F exceeds k.*

To highlight the relevance of Theorem 2, recall Proposition 1 also holds for chordal AFs. In particular, in this fragment, both treewidth and minimum *ACYC*-backdoor size can be arbitrarily larger than the *ACYC*-backdoor-treewidth.

Theorem 2. *Given a chordal AF F and a parameter k, it is fixed-parameter tractable to either return an ACYC-backdoor of width $\leq k$ or correctly conclude that the ACYC-backdoor-treewidth of F exceeds k.*

5 Utilizing Small Backdoor-Treewidth

In the following we present FPT results for AFs parameterized by backdoor-treewidth. This is established by means of dynamic programming (DP) algorithms for AFs that explicitly utilize backdoor-treewidth. In such DP algorithms, extensions are often times captured by *colorings* that are computed for each bag of a TD in post-order (bottom-up). These colorings give rise to invariants that need to be fulfilled for every node. Maintaining colorings is similar to existing DP algorithms [20] for treewidth, but for backdoor-treewidth one also needs to take care of tractable components induced by leaf nodes.

Next, we show that reasoning on AFs is fixed-parameter tractable parameterized for backdoor-treewidth to *ACYC* or *NOEVEN*, if a backdoor of minimum width is given.

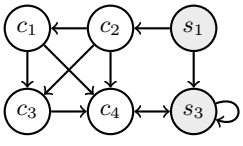
Theorem 3. *Reasoning and counting on AFs F in stable and complete semantics is fixed-parameter tractable when parameterized by $\mathtt{btw}_{\mathcal{C}}(F)$ for $\mathcal{C} \in \{ACYC, NOEVEN\}$ if a respective backdoor is given.*

This result is established by adapting the existing DP algorithm in consideration of our torso graphs. In particular, we consider the tractable S-components to be "attached" to the sub-problems in the leaves of a torso tree decomposition. Then, tractability is preserved by applying the backdoor algorithm approach in the leaf nodes as in related work [15]. To this end, for a set $S \subseteq A$, a tree decomposition $(\mathcal{T}, \mathcal{X})$ of torso graph G_F^S, and $t \in \mathcal{T}$, let $X_{\geq t}$ be the union of all bags $X_s \in \mathcal{X}$ and sets *components*(s) for every node s that occurs in the subtree of $\mathcal{T}$ rooted at t. Further, $X_{>t}$ shall denote $X_{\geq t} \setminus X_t$.

Observe that for leaf nodes t of $\mathcal{T}$, we have $X_{>t} =$ *components*(t). Consequently, in the leaf nodes of $\mathcal{T}$, we guess a coloring on the backdoor-arguments X_t in the bag and check whether this coloring yields $X_{>t}$-restricted stable/complete extensions, cf. Definition 1. Note that as X_t for a leaf node $t \in \mathcal{T}$ is a backdoor set, this check can be performed in polynomial time. This follows with slight adaptations from the respective results by Dvořák et al. [15]. The remainder of the adapted DP algorithms proceeds as in the original algorithms for stable/complete extensions. In particular, every node maintains those colorings yielding $X_{>t}$-restricted stable/complete sets. Then, one can extend the correctness proofs for the modified algorithms returning valid colorings (see appendix for details).

Example 1 ctd. We evaluate the leaf node $3'$ of the given TD of the torso $G_F^{B'}$ for stable semantics on the AF $F_{\geq 3}$ (see below). Following the backdoor approach [15], we guess colors on the backdoor arguments $B' = \{s_1, s_3\}$ (gray background in illustration below), such that the set of arguments colored *in* is conflict-free in $F_{\geq 3}$, and each argument that is attacked by an *in*-argument is colored *def* ("defeated").

$F_{\geq 3}$:



Then, for each such guessed coloring λ we compute the *propagation* λ^* on the remaining non-backdoor arguments $x \in \mathit{components}(3') = \{c_1, c_2, c_3, c_4\}$ (white background in illustration above): for stable semantics it suffices to guess whether an argument is *in* or *def*. Assume we guess $\lambda(s_1) = \mathit{in}$, $\lambda(s_3) = \mathit{def}$. Consequently, by applying the propagation rules, we obtain $\lambda^*(c_2) = \mathit{def}$, $\lambda^*(c_1) = \mathit{in}$, and $\lambda^*(c_3) = \lambda^*(c_4) = \mathit{def}$. This corresponds to the B'-restricted stable set $\{s_1, c_1\}$. Now assume we guess $\lambda(s_1) = \lambda(s_3) = \mathit{def}$. We obtain $\lambda^*(c_2) = \mathit{in}$, and $\lambda^*(c_1) = \lambda^*(c_3) = \lambda^*(c_4) = \mathit{def}$. This corresponds to the B'-restricted stable set $\{c_2\}$.

Limitations of Backdoor-Treewidth. The backdoor-treewidth approach, however, does not always work. In the following, we present several cases that do not admit fixed-parameter tractability when parameterized by backdoor-treewidth (under standard assumptions of complexity theory). In particular, this holds for the tractable fragments *BIP* and *SYM* (i.e., even though Theorem 1 guarantees us *finding SYM*-backdoors is fixed-parameter tractable, *reasoning* on an AF of constant *SYM*-backdoor-treewidth remains hard).

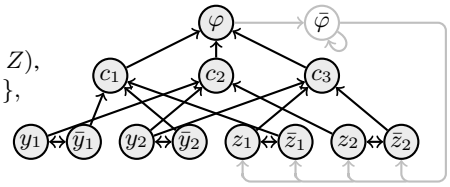
Proposition 3. *Reasoning on AFs F with $\mathtt{btw}_{\mathcal{C}}(F) = 0$ in $\sigma \in \{\mathit{adm}, \mathit{com}, \mathit{pref}, \mathit{stb}\}$ and $\mathcal{C} \in \{\mathit{SYM}, \mathit{BIP}\}$ remains* NP/coNP*-hard.*

Several fragments have the property that the otherwise Π_2^{P}-hard skeptical acceptance problem becomes coNP-easy, such as (a) odd-cycle-freeness [13], (b) no even-cycles of length ≥ 4 [18] (subsuming the class of AFs that have no cycles longer than 3 arguments). We will denote these by (a) *ODDFREE* and (b) *EVENFREE-4*, respectively.

Proposition 4. *Skeptical acceptance for preferred semantics is Π_2^{P}-hard even for AFs F with $\mathtt{btw}_{\mathcal{C}}(F) = 0$ for $\mathcal{C} \in \{\mathit{ODDFREE}, \mathit{EVENFREE\text{-}4}\}$.*

Proof. Dvořák et al. [16] showed Π_2^{P} hardness for the problem $\mathit{Skept}_{\mathit{pref}}$ for AFs F with $\mathtt{bd}_{\mathit{ODDFREE}}(F) = 1$ in a different context. The same reduction F_Φ (see example below) proves hardness for *EVENFREE-4* for $\mathtt{bd}_{\mathit{EVENFREE\text{-}4}}(F) = 1$.

F_Φ for
$\Phi = \forall Y \exists Z \varphi(Y, Z)$,
$\varphi = \{\{\bar{y}_1, \bar{y}_2, \bar{z}_1\},$
$\{y_1, y_2, z_2\},$
$\{y_2, z_1, \bar{z}_2\}\}$.



It suffices to remove the highlighted argument $\bar{\varphi}$, hence, $\mathtt{bd}_{\mathcal{C}}(F_\Phi) = 1$ and also $\mathtt{btw}_{\mathcal{C}}(F_\Phi) = 0$. □

6 Systems and Benchmarks

We implemented both a system to find the exact (e.g., minimal) width of all *ACYC*-backdoors, as well as an argumentation problem solver using the backdoor-treewidth approach. In the latter, we solve the "Count Extensions"-problem in semantics $\sigma \in$

$\{stb, adm\}$, where given a framework F we ask for the number of extensions $|\sigma(F)|$. This problem is gaining importance in the community, and has recently been added to the ICCMA competition [28].

Benchmark Hardware. All our solvers ran on a cluster consisting of 12 servers. Each of these servers is equipped with two Intel Xeon E5-2650 CPUs, consisting of 12 physical cores that run at 2.2 GHz clock speed and have access to 256 GB shared main memory (RAM). Results are gathered on Ubuntu 16.04.1 LTS powered on kernel 4.4.0-139 with hyperthreading disabled using version 3.7.6 of Python3.

Benchmark Instances. As we focused on extreme values for backdoor size and treewidth, in addition to using instances provided by the ICCMA competition, we generated AFs to range from small to high backdoor sizes/treewidths. Then, we performed our experiments on a variety of composite frameworks. This is motivated by the modular nature of many frameworks that arise from applications. In particular, we combined instances provided by the ICCMA competition with frameworks designed to range from small to large backdoor-sizes and treewidths.

For scenario (A), we generated 960 instances that contain dense, directed subgraphs (high treewidth) and sparse subgraphs with many cycles (high backdoor size). For scenario (B), we combined small ICCMA instances with the generated frameworks from (A), resulting in 1134 instances:
(B1) 378 combinations of only ICCMA instances,
(B2) 378 combinations of only generated instances,
(B3) 378 combinations of ICCMA *and* generated instances.

Finding *ACYC*-Backdoors of Minimal Width

We propose a SAT encoding that, given an AF F and an integer k, produces a propositional formula $E(F, k)$ that is satisfiable if and only if $\texttt{btw}(F) \leq k$. We construct the formula in three steps: (i) we encode the definition of an *ACYC*-backdoor, (ii) we derive the corresponding torso graph, and (iii) we restrict the treewidth of the torso graph [31]. We then find the $\texttt{btw}(F)$ of an instance by incrementally increasing k until the SAT solver finds the formula satisfiable. For the 1134 instances of scenario B we computed $\texttt{btw}(F)$ using our SAT encoding. For 611 of the 1134 instances, that is more than half, $\texttt{btw}(F) < \texttt{tw}(F)$, and for most of them (519 out of 611), even $\texttt{btw}(F) \leq \texttt{tw}(F)/2$. Details are given in the appendix.

Utilizing Small Backdoor-Treewidth in Practice

We refer to the implemented system by argBTW[1]. It can easily be extended to the credulous/skeptical acceptance-problem, the existence of extensions-problem, etc. Our system argBTW uses *decomposition-guided reductions* to propositional satisfiability [22]. Such a reduction for argumentation reduces the respective argumentation problem to propositional logic such that the treewidth of the argumentation problem is (linearly)

[1]Find argBTW and benchmarks at github.com/mk-tu/argBTW.

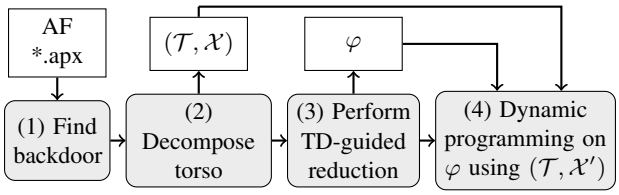


Figure 2: Flowchart of the implemented argBTW system.

preserved. This way, we can adapt the existing NestHdb system [26] for propositional logic in order to efficiently implement dynamic programming for argumentation.

As an input, we get an AF in `apx` format. Then, four steps are performed (illustrated in Figure 2): (1) Find a suitable backdoor to *ACYC*. (2) Using this backdoor, compute the torso graph and decompose it into a suitable TD $(\mathcal{T}, \mathcal{X})$ of small width. (3) Perform tree decomposition guided reduction and get a propositional formula φ characterizing extensions of the AF. (4) Following the structure given by $\mathcal{T}$, we apply a dynamic programming algorithm on φ to determine the number of models of φ, i.e., the number of extensions.

In the previous section, we discussed an approach for computing the backdoor-treewidth exactly. For practical purposes and in order solve reasonably-sized instances efficiently, we rely on heuristics, i.e., Steps (1) and (2) as described above do not use exact methods. For (1) we apply an *Answer Set Programming (ASP)* encoding: for (2) we employ the htd system [1].

For Step (1) we allotted at most 30 seconds and Step (2) is finished within a second per instance, due to the decomposer htd being reasonably fast. Step (4) also includes a simple preprocessing phase, which aims at simplifying instances quickly via, e.g., techniques like unit propagation.

Empirical Evaluation

The performed experiments aimed at identifying graph structures where the backdoor-treewidth approach is particularly beneficial. To this end, we considered two scenarios:

(A) We compared the backdoor-treewidth approach to an backdoor-only and a treewidth-only approach.

(B) As the backdoor-treewidth approach turned out to be the fastest of the three, we conducted experiments where we compared this method on counting extensions against an ASP approach (ASPARTIX, 17).

For Scenario (A), we are interested in comparing our technique based on backdoor-treewidth with the special (degenerated) cases of treewidth and backdoor size. Then, the conducted experiments of Scenario (B) aim at examining the runtime behavior of the "Count Extensions"-problem. This problem is quite general and can very easily be adapted to decide credulous/skeptical acceptance of an argument, existence of a (non-empty) extension, and other well established argumentation problems. In our experiments, we mainly compare wall clock time and number of solved instances over

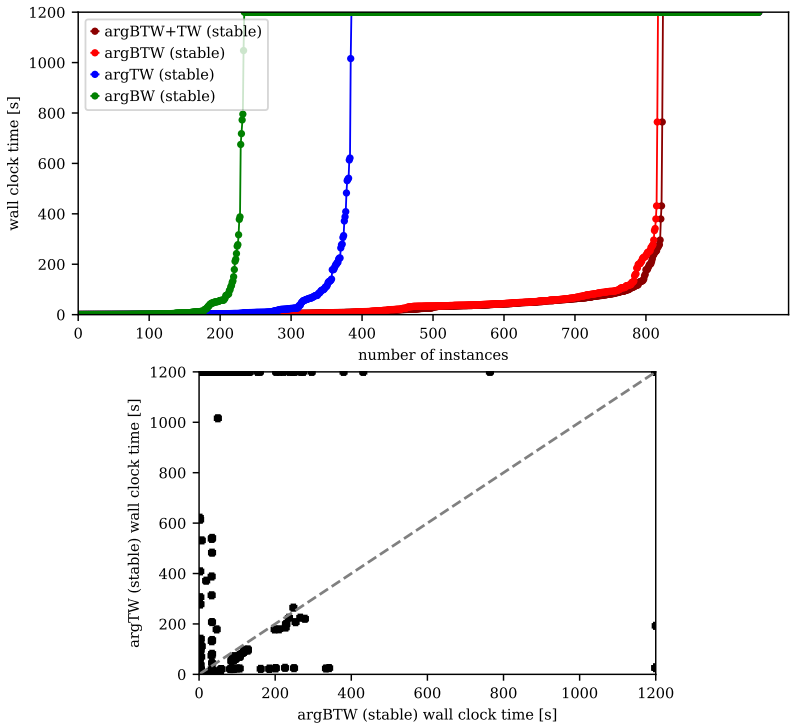


Figure 3: Performance comparison of argBTW, argTW and argBW for Scenario (A). The cactus plot (left) shows the number of solved instances (x-axis) that is sorted in ascending order for each approach individually according to runtime. The scatter plot (right) depicts a one-to-one comparison between the runtime (in seconds) of every instance for BTW and TW.

the best of two runs for every solver and instance, where each run is allowed to use up to 1200s of wall clock time and 16GB of main memory (RAM).

The goal of the two scenarios (A) and (B), is to empirically confirm the following hypotheses.

(H1$_A$) There are instances where backdoor-treewidth based solvers outperform (tree)-width and backdoor approaches.

(H2$_A$) In practice, the obtained backdoor-treewidths are often smaller than the computed widths and backdoor sizes.

(H$_B$) For counting extensions, the backdoor-treewidth based solver and suitable portfolio configurations combining argBTW and argTW often outperform existing approaches based on ASP.

Benchmarked Systems. In our experiments, we mainly compare the performance of the following configurations.

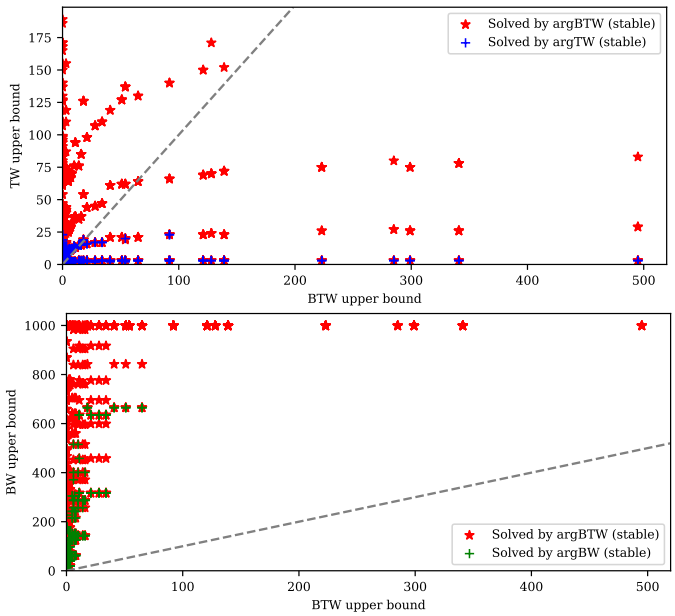


Figure 4: Measures among solved instances of Scenario (A). We compare (left) upper bounds of backdoor-treewidth (BTW) against treewidth (TW) as well as (right) upper bounds of backdoor-treewidth (BTW) against backdoor-size (BW).

- clingo[2] (stable) and clingo (admissible): these configurations are based on clingo version 5.4.0 and we used arguments "-q" and "-n 0" when counting answer sets. On top we used standard encodings for admissible and stable semantics of ASPARTIX [17].
- argBTW (stable) and argBTW (admissible): see above.
- argBW (stable) and argBW (admissible): this is a special case of argBTW, where instead of computing and decomposing a torso of an AF, after finding a backdoor B we solve the instance via the backdoor-only approach.
- argTW (stable) and argTW (admissible): in this special case of argBTW, we take all arguments A of an AF F as a backdoor, obtaining the undirected graph U^F as a torso, which is then solved with the treewidth-only approach.
- argBTW+TW (stable) and argBTW+TW (admissible): This is a portfolio that emerged from our study. First, 200s are used for solving via argTW and then if unsuccessful, the remaining 1000s are put into argBTW.

Results of Scenario (A). An overview of the results for Scenario (A) is depicted in Figure 3. Figure 3 (left) shows a cactus plot over the different configurations of our solver for computing stable extensions. Such a cactus plot depicts for each configuration the number of solved instances (x-axis) according to their runtime (y-axis) in

[2]ASP solver, see https://potassco.org/

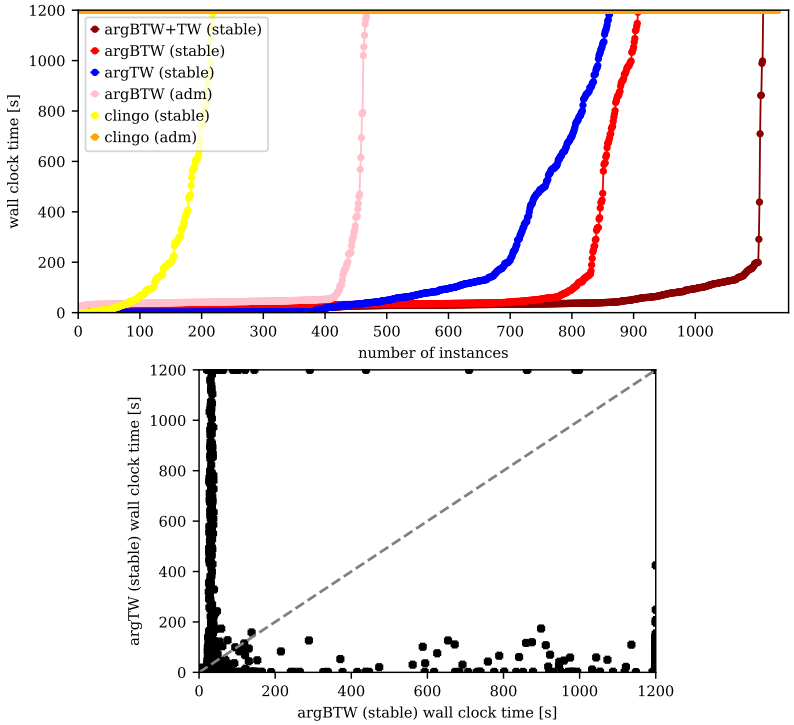


Figure 5: Performance of compared solvers for Scenario (B). The cactus plot (left) shows the BTW approach for admissible and stable semantics. The scatter plot (right) allows a one-to-one comparison of solved instances for BTW vs. TW approach.

ascending order. Consequently, one sees that overall the backdoor-treewidth based approach argBTW outperforms the other configurations. The picture is similar for admissible extensions (not shown in the plot). To enable a one-to-one comparison, Figure 3 (right) depicts a scatter plot, where the runtime of each instance is compared between argBTW (x-axis) and the second best configuration of Figure 3 (left), namely argTW (y-axis). The argBTW approach is often faster and solves more instances, thereby confirming Hypothesis ($H1_A$), however, there are also some instances below the dashed diagonal, which indicates instances, where argTW is faster. This scatter plot gives rise to the portfolio configuration argBTW+TW in order to further improve the performance, since almost all of these dots can be solved by argTW in a runtime below or even well below 200 seconds. While the runtime benefits of argBTW are certainly interesting, in order to evaluate Hypothesis ($H2_A$), we analyze the obtained upper bounds of backdoor-treewidths, treewidths and backdoor sizes. The values of these measures among all solved instance are compared in Figure 4, where (left) compares the backdoor-treewidth upper bounds (x-axis) to the used widths (y-axis) and (right) compares backdoor-treewidth upper bounds (x-axis) with used backdoor sizes (y-axis). These scatter plots allow a one-to-one comparison of these measured values for each instance solved by the respective configuration. Overall, one can observe that while

there are some instances where we obtain larger backdoor-treewidth than treewidth upper bounds (cf. dots below the diagonal of Figure 4 (left)), the obtained backdoor size is always dominated by backdoor-treewidth upper bounds as shown in Figure 4 (right). Both observations confirm Hypothesis $H2_A$.

Results of Scenario (B). We took the best configurations of Scenario (A) in order to count extensions over instances of different kind. Figure 5 presents the overall results of this experiment, where in (left) we show a cactus plot comparing our approach with the treewidth-based technique as well as solutions based on logic programming (ASP). Further, Figure 5 (right) completes this analysis by a one-to-one comparison of argBTW vs. argTW and explains why the portfolio argBTW+TW is promising. In particular, this plot also reveals that there is still further potential for improving our heuristics on approximating decent backdoor-treewidth upper bounds. We provide detailed results in Table 1, revealing that the portfolio argBTW+TW shows indeed a significant performance increase compared to argBTW. Overall, we can confirm Hypothesis H_B for counting extensions.

7 Conclusion

In this work, we introduced the parameter backdoor-treewidth for argumentation, which dominates the well established parameters treewidth and minimum backdoor size. We gave algorithms to compute backdoors of width $\leq k$ that establish fixed-parameter tractability (w.r.t. k) for the fragment *SYM* in the general case, and for the fragment *ACYC* for chordal AFs. Moreover, we showed fixed-parameter tractability for solving several problems associated with AFs if a suitable backdoor to one of the fragments *ACYC* or *NOEVEN* is given. On the other hand, we established computational hardness for these problems in other fragments, effectively pointing out limitations of the new backdoor-treewidth approach. Finally, we conducted systematic experiments, empirically evaluating the power of the newly introduced parameter. Here, we presented systems for both finding the exact backdoor-treewidth (as well as the associated backdoor set and a suitable tree decomposition of the torso), and a composite system for solving argumentation problems. We found several graph structures where backdoor-treewidth approach outperforms its "parent"-parameters treewidth and minimum backdoor size. However, as Figure 4 suggests, the hereby used heuristics for finding adequate backdoor sets can still be improved, i.e., there is further open potential for estimating tree decompositions of decent backdoor-treewidth. Further future work regards investigations for other semantics and fragments, as well as the connection to structured argumentation.

Acknowledgments

This research has been supported by the Vienna Science and Technology Fund (WWTF) through project ICT19-065, and by the Austrian Science Fund (FWF) through projects P30168, P32441, P32830, W1255, and Y698. Part of the project was carried out while

Solver	$\sum$	Width Range				Time[h]
		max(width)	0-5	5-10	>10	
		B1				
argTW (stable)	**378**	**16.0**	**29**	**163**	**186**	**0.18**
argBTW (stable)	279	15.0	**29**	158	92	43.45
argBTW (adm)	37	14.0	23	12	2	116.72
clingo (stable)	31	10.0	17	14	0	117.54
clingo (adm)	0	0	0	0	0	126.0
		B2				
argBTW (stable)	**377**	**3.0**	**377**	0	0	**3.72**
argBTW (adm)	**377**	**3.0**	**377**	0	0	4.71
argTW (stable)	184	**3.0**	184	0	0	94.5
clingo (stable)	129	**3.0**	129	0	0	91.03
clingo (adm)	0	0	0	0	0	126.0
		B3				
argTW (stable)	**300**	**16.0**	36	102	**162**	**31.79**
argBTW (stable)	252	15.0	**46**	**133**	73	52.19
clingo (stable)	60	**16.0**	28	23	9	109.33
argBTW (adm)	54	11.0	44	9	1	110.67
clingo (adm)	0	0	0	0	0	126.0
		Σ				
argBTW+TW (stable)	**1110**	**16.0**	**452**	**296**	**362**	**19.99**
argBTW (stable)	**908**	15.0	**452**	291	165	99.36
argTW (stable)	862	**16.0**	249	265	348	126.47
argBTW (adm)	468	14.0	444	21	3	232.1
clingo (stable)	220	**16.0**	174	37	9	317.9
clingo (adm)	0	0	0	0	0	378.0

Table 1: Detailed benchmark data for Scenario (B), where we show for each configuration the number of solved instances, grouped by respective width upper bounds, as well as the overall runtime (timeouts account for 1200s).

Hecher, Schidler, and Szeider were visiting the Simons Institute for the Theory of Computing. Hecher is also affiliated with the university of Potsdam, Germany.

References

[1] Michael Abseher, Nysret Musliu, and Stefan Woltran. htd - A free, open-source framework for (customized) tree decompositions and beyond. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua,*

Italy, June 5-8, 2017, Proceedings, volume 10335 of *LNCS*, pages 376–386. Springer, 2017. doi: 10.1007/978-3-319-59776-8_30.

[2] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *Siam Journal of Discrete Mathematics*, 8(2): 277–284, 1987.

[3] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.

[4] Pietro Baroni, Paul E. Dunne, and Massimiliano Giacomin. On extension counting problems in argumentation frameworks. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo R. Simari, editors, *Proceedings of the 3rd Conference on Computational Models of Argument (COMMA 2010)*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 63–74, Desenzano del Garda, Italy, September 8-10 2010. IOS Press. ISBN 978-1-60750-618-8.

[5] Hans L. Bodlaender. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'05)*, volume 3381 of *LNCS*, pages 1–16. Springer, 2005.

[6] Martin Caminada and Dov M. Gabbay. A logical account of formal argumentation. *Studia Logica*, 93(2):109–145, 2009. ISSN 0039-3215.

[7] Federico Cerutti, Sarah A. Gaggl, Matthias Thimm, and Johannes P. Wallner. Foundations of implementations for formal argumentation. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14, pages 688–767. College Publications, 2018. also appears in IfCoLog Journal of Logics and their Applications 4(8):2623–2706.

[8] Günther Charwat. Tree-decomposition based algorithms for abstract argumentation frameworks. Master's thesis, TU Wien, 2012. URL http://permalink.obvsg.at/AC07812654.

[9] Sylvie Coste-Marquis, Caroline Devred, and Pierre Marquis. Symmetric argumentation frameworks. In Lluis Godo, editor, *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005)*, volume 3571 of *LNCS*, pages 317–328. Springer, 2005. ISBN 3-540-27326-3.

[10] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

[11] Paul E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artif. Intell.*, 171(10-15):701–729, 2007.

[12] Paul E. Dunne and Trevor J. M. Bench-Capon. Complexity and combinatorial properties of argument systems. Technical report, Dept. of Computer Science, University of Liverpool, 2001.

[13] Paul E. Dunne and Trevor J. M. Bench-Capon. Coherence in finite argument systems. *Artif. Intell.*, 141(1/2):187–203, 2002.

[14] Wolfgang Dvořák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. *FLAP*, 4(8), 2017. URL http://www.collegepublications.co.uk/downloads/ifcolog00017.pdf.

[15] Wolfgang Dvořák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. *Artificial Intelligence*, 186(0):157–173, 2012. ISSN 0004-3702. doi: 10.1016/j.artint.2012.03.002. URL http://www.sciencedirect.com/science/article/pii/S0004370212000239.

[16] Wolfgang Dvořák, Matti Järvisalo, Johannes Peter Wallner, and Stefan Woltran. Complexity-sensitive decision procedures for abstract argumentation. *Artificial Intelligence*, 206(0):53 – 78, 2014. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/j.artint.2013.10.001.

[17] Wolfgang Dvořák, Sarah Alice Gaggl, Anna Rapberger, Johannes Peter Wallner, and Stefan Woltran. The ASPARTIX system suite. In *COMMA*, volume 326 of *Frontiers in Artificial Intelligence and Applications*, pages 461–462. IOS Press, 2020.

[18] Wolfgang Dvořák, Matthias König, and Stefan Woltran. On the complexity of preferred semantics in argumentation frameworks with bounded cycle length. In Meghyn Bienvenu, Gerhard Lakemeyer, and Esra Erdem, editors, *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, pages 671–675, 2021. doi: 10.24963/kr.2021/67. URL https://doi.org/10.24963/kr.2021/67.

[19] Wolfgang Dvořák, Markus Hecher, Matthias König, André Schidler, Stefan Szeider, and Stefan Woltran. Tractable abstract argumentation via backdoor-treewidth. In *36th AAAI Conference on Artificial Intelligence*, 2022. To appear.

[20] Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artif. Intell.*, 186:1 – 37, 2012. ISSN 0004-3702. doi: 10.1016/j.artint.2012.03.005.

[21] Johannes K. Fichte, Markus Hecher, and Arne Meier. Counting complexity for reasoning in abstract argumentation. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 2827–2834. AAAI Press, 2019. ISBN 978-1-57735-809-1.

[22] Johannes Klaus Fichte, Markus Hecher, Yasir Mahmood, and Arne Meier. Decomposition-guided reductions for argumentation and treewidth. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1880–1886. ijcai.org, 2021. doi: 10.24963/ijcai.2021/259. URL https://doi.org/10.24963/ijcai.2021/259.

[23] Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining treewidth and backdoors for CSP. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 36:1–36:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi: 10.4230/LIPIcs.STACS.2017.36. URL https://doi.org/10.4230/LIPIcs.STACS.2017.36.

[24] Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Backdoor treewidth for SAT. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017*, volume 10491 of *LNCS*, pages 20–37. Springer, 2017. doi: 10.1007/978-3-319-66263-3_2. URL https://doi.org/10.1007/978-3-319-66263-3_2.

[25] Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction, and database problems. *The Computer Journal*, 51(3):303–325, 2008.

[26] Markus Hecher, Patrick Thier, and Stefan Woltran. Taming high treewidth with abstraction, nested dynamic programming, and database technology. In *Theory and Applications of Satisfiability Testing - SAT 2020*, volume 12178 of *LNCS*, pages 343–360. Springer, 2020.

[27] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. ISBN 3-540-58356-4. doi: 10.1007/BFb0045375.

[28] Jean-Guy Mailly, Emmanuel Lonca, Jean-Marie Lagniez, and Julien Rossit. International Competition on Computational Models of Argumentation (ICCMA) 2021, 2021. Available at: https://www.argumentationcompetition.org/2021.

[29] Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In *Principles and Practice of Constraint Programming*, pages 531–548. Springer International Publishing, 2014.

[30] Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[31] Marko Samer and Helmut Veith. Encoding treewidth into SAT. In *Theory and Applications of Satisfiability Testing - SAT 2009. Proceedings*, volume 5584 of *LNCS*, pages 45–50. Springer, 2009.

A Proofs of Section 4

Proposition 2. *Deciding whether a given AF F has an ACYC-backdoor of width $\leq k$ is* NP*-complete (if k is part of the input).*

Proof. NP-membership is clear, as a given tree decomposition of a torso can be verified in polynomial time. To establish hardness we can just add self-attacks to every argument in F—the only *ACYC*-backdoor for the resulting AF F' is the set of all arguments. Hence, the backdoor-treewidth of F' is the treewidth of F. It is well known that deciding whether an undirected graph has treewidth $\leq k$ is NP-complete [2]. □

Let $\mathcal{V}$ be a set of variables and $\mathcal{D}$ a finite set of values. A *constraint of arity ρ over $\mathcal{D}$*, $\rho \geq 0$, is a pair $c = (s, r)$ where $s = (x_1, \ldots, x_\rho)$ is a sequence of variables from $\mathcal{V}$ and $r \subseteq \mathcal{D}^\rho$ is a ρ-ary relation. The set $\text{var}(s) = \{x_1, \ldots, x_\rho\}$ is the *scope* of c. For $\rho > 0$ let $\mathcal{D}^\rho$ denote the full ρ-ary relation over $\mathcal{D}$, let $\bot_\rho$ denote the empty ρ-ary relation, and $\top$ the 0-ary relation with the empty tuple.

Let $\alpha : X \to \mathcal{D}$ be an assignment. For a ρ-ary constraint $c = (s, r)$ with $s = (x_1, \ldots, x_\rho)$ and $r \subseteq \mathcal{D}^\rho$, we denote by $c|_\alpha$ the constraint (s', r') that we obtain from c as follows. We obtain r' from r by (i) deleting all tuples $(d_1, \ldots, d_\rho)$ from r for which there is some $1 \leq i \leq \rho$ such that $x_i \in X$ and $\alpha(x_i) \neq d_i$, and (ii) removing from all remaining tuples all coordinates d_i with $x_i \in X$. We obtain s' from s by deleting all variables x_i with $x_i \in X$. If $\text{var}(c) \subseteq X$ then $\text{var}(c|_\alpha) = \emptyset$ and $\text{rel}(c|_\alpha) \in \{\top, \bot_0\}$. An instance of the Constraint Satisfaction Problem (CSP) or *CSP instance*, is a finite set of constraints over a domain $\mathcal{D}$. For a CSP instance $\mathcal{I}$ we define $\mathcal{I}|_\alpha = \{\, c|_\alpha : c \in \mathcal{I} \,\}$.

A *constraint language* Γ over a domain $\mathcal{D}$ is a set of relations (of possibly various arities) over $\mathcal{D}$. By $\mathcal{C}(\Gamma)$ we denote the set of all CSP instances $\mathcal{I}$ with $\{\, \text{rel}(c) : c \in \mathcal{I} \,\} \subseteq \Gamma$.

Let Γ be a constraint language over $\mathcal{D}$ and $\mathcal{I}$ a CSP instance over $\mathcal{D}$. A set $X \subseteq \text{var}(\mathcal{I})$ is a *strong Γ-backdoor* if for each assignment $\alpha : X \to \mathcal{D}$ it holds that $\mathcal{I}|_\alpha \in \mathcal{C}(\Gamma)$.

The *incidence graph* $G(\mathcal{I})$ of a CSP instance $\mathcal{I}$ is the bipartite graph whose vertices correspond to the variables and constraints of $\mathcal{I}$, and where vertices corresponding to a variable x and a constraint c are adjacent if and only if $x \in \text{var}(c)$. For a subset $X \subseteq \text{var}(\mathcal{I})$ we denote by $\mathsf{Torso}_\mathcal{I}(X)$ the graph with vertex set X which contains the edge uv if $u \neq v$ and both have a neighbor in the same connected component of $G(\mathcal{I}) - X$.

Theorem 4 (Ganian et al. 23)**.** *Let Γ be any finite constraint language over $\mathcal{D}$ containing the relation $\mathcal{D}^2$. Given a CSP instance $\mathcal{I}$ and a parameter k, it is fixed-parameter tractable to either return a set X of variables such that X is a strong backdoor of $\mathcal{I}$ to $\mathcal{C}(\Gamma)$ where the treewidth of $\mathsf{Torso}_\mathcal{I}(X)$ is at most k, or correctly conclude that no such set exists.*

Let $F = (A, R)$ be an AF and $C = \{a_1, \ldots, a_\ell\} \subseteq A$ a set of arguments. C is a *cycle of length ℓ in F* if R contains all the attacks (a_i, a_{i+1}), $1 \leq i \leq \ell - 1$, as well as the attack (a_ℓ, a_1); this includes the case where $\ell = 1$ and $(a_1, a_1) \in R$. We call F *chordal* if each cycle C of F contains a cycle C' of length ≤ 3. We would like to point out that the ordering of the arguments in C' may be different than the ordering in C.

The significance of the next theorem lies in the fact that there are chordal AFs with constant *ACYC*-backdoor treewidth that have arbitrarily large treewidth and *ACYC*-backdoor size.

Theorem 2. *Given a chordal AF F and a parameter k, it is fixed-parameter tractable to either return an ACYC-backdoor of width $\leq k$ or correctly conclude that the ACYC-backdoor treewidth of F exceeds k.*

Proof. Let Γ_0 be the constraint language over $\mathcal{D} = \{0, 1\}$ consisting of the relations $\top$, $\bot_0$, $\{0\}$, $\{1\}$, $\{0, 1\}$, and $\mathcal{D}^2$. Given a chordal AF $F = (A, R)$, we construct a CSP instance $\mathcal{I}_F$ with $\text{var}(\mathcal{I}) = A$ as follows. We consider all cycles C of F of length ≤ 3. For each such cycle C we add a constraint c to $\mathcal{I}_F$ with $\text{var}(c) = C$ and

$$\text{rel}(c) = \begin{cases} \bot_1 & \text{if } |C| = 1; \\ \{(0,0),(1,1)\} & \text{if } |C| = 2; \\ \mathcal{D}^3 & \text{if } |C| = 3. \end{cases}$$

For each attack $(a, b) \in R$ we add a constraint c with $\text{var}(c) = \{a, b\}$ and $\text{rel}(c) = \mathcal{D}^2 \in \Gamma_0$.

Claim 1: $X \subseteq A$ is an ACYC-backdoor of F if and only if X is a strong Γ_0 backdoor of $\mathcal{I}_F$. To prove the claim, let X be an *ACYC*-backdoor of F and let $\alpha : X \to \mathcal{D}$. Assume to the contrary that $\mathcal{I}|_\alpha \notin \mathcal{C}_{\Gamma_0}$, i.e., there is some $c^* \in \mathcal{I}$ such that $\text{rel}(c^*) \notin \Gamma_0$. However, Γ_0 is closed under partial assignments (i.e., for each constraint c with $\text{rel}(c) \in \Gamma_0$ and each assignment α we have $\text{rel}(c|_\alpha) \in \Gamma_0$), c^* must originate from some constraint that we added because of a cycle. However, for each constraint c that we added because of a cycle, and each partial assignment $\alpha : X \to \mathcal{D}$ with $X \cap \text{var}(c) \neq \emptyset$, we have $\text{rel}(c|_\alpha) \in \Gamma_0$. Hence $c^* = c$ for constraint that we added for a cycle C. However, since $X \cap C = \emptyset$, C is still a cycle in $F - X$, hence X is not an *ACYC*-backdoor of $\mathcal{I}_F$, a contradiction. See Figure 6 for an illustration.

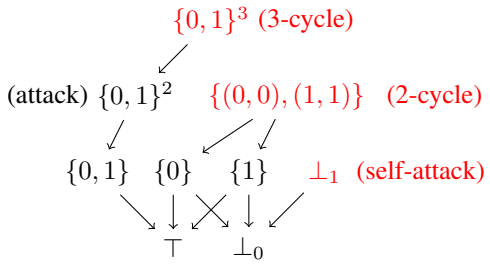


Figure 6: Illustration for the proof of Theorem 2. $r \to r'$ indicates that by assigning one variable from a constraint c with $\text{rel}(c) = r$ we can obtain a constraint c' with $\text{rel}(c') = r'$. Relations in Black belong to Γ_0, relations in Red do not.

For the converse direction, let X be a strong Γ_0-backdoor of $\mathcal{I}_F$. Assume to the contrary that F contains a cycle C with $C \cap X = \emptyset$. Since F is chordal, there is some cycle $C' \subseteq C$ of length ≤ 3 in F. Let c be the constraint added to $\mathcal{I}_F$ because of C'.

We have $\text{rel}(c) \notin \Gamma_0$, and since $X \cap C' = \emptyset$, $c \in \mathcal{I}|_\alpha$ for any $\alpha : \mathcal{I} \to \mathcal{D}$. Hence X isn't a strong Γ_0-backdoor of $\mathcal{I}_F$, a contradiction. Thus Claim 1 holds.

Claim 2: the width of $X \subseteq A$ for F equals the treewidth of $\mathsf{Torso}_{\mathcal{I}_F}(X)$. Claim 2 follows from the fact that the graphs $\mathsf{Torso}_{\mathcal{I}}(X)$ and G_F^X are identical.

Since the construction of $\mathcal{I}_F$ from F can be carried out in polynomial time (in $O(|F|)^3$), the theorem follows via Claims 1 and 2 from Theorem 4. □

Theorem 1. *Given an AF F and a parameter k, it is fixed-parameter tractable to either returns a SYM-backdoor of width $\leq k$ or correctly conclude that the SYM-backdoor treewidth of F exceeds k.*

Proof. As in the proof of Theorem 2, we will reduce the problem to the problem of finding a strong Γ_0-backdoor in a CSP instance $\mathcal{I}_F$. We construct a CSP instance $\mathcal{I}_F$ from a given AF $F = (A, R)$ with $\text{var}(\mathcal{I}) = A$. For each pair $(a, b) \in R$, we add a constraint c to $\mathcal{I}_F$ where $\text{var}(c) = \{a, b\}$ and

$$\text{rel}(c) = \begin{cases} \{(0,0),(1,1)\} \notin \Gamma_1 & \text{if } (b,a) \notin R; \\ \mathcal{D}^2 \in \Gamma_1 & \text{if } (b,a) \in R \;\&\; a \neq b; \\ \bot_1 \notin \Gamma_1 & \text{if } a = b. \end{cases}$$

Claim 1: $X \subseteq A$ is a SYM-backdoor of F if and only if X is a strong Γ_1 backdoor of $\mathcal{I}_F$. The claim follows by observing that for destroying the forbidden configurations in F (self-attacks or non-symmetric attacks) is equivalent with destroying the forbidden constraints in $\mathcal{I}_F$ (constraints with the relations $\bot_1$ and $\{(0,0),(1,1)\}$), see Figure 7.

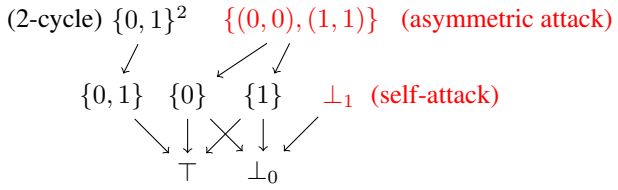


Figure 7: Illustration for the proof of Theorem 1.

Claim 2: the width of $X \subseteq A$ for F equals the treewidth of $\mathsf{Torso}_{\mathcal{I}_F}(X)$. Claim 2 follows, as above, from the fact that the graphs $\mathsf{Torso}_{\mathcal{I}}(X)$ and G_F^X are identical. The conjunction of both claims, together with the fact that we can construct $\mathcal{I}_F$ from F in polynomial time, allows us to obtain the theorem from Theorem 4. □

B Proofs of Section 5

Proposition 3. *Reasoning on AFs F with $\mathtt{btw}_{\mathcal{C}}(F) = 0$ in $\sigma \in \{adm, com, pref, stb\}$ and $\mathcal{C} \in \{SYM, BIP\}$ remains* NP/coNP*-hard.*

Proof. The respective problems are hard even for AFs F with a minimal $\mathcal{C}$-backdoor size of 1 [15], which implies $\mathtt{btw}_{\mathcal{C}}(F) = 0$. □

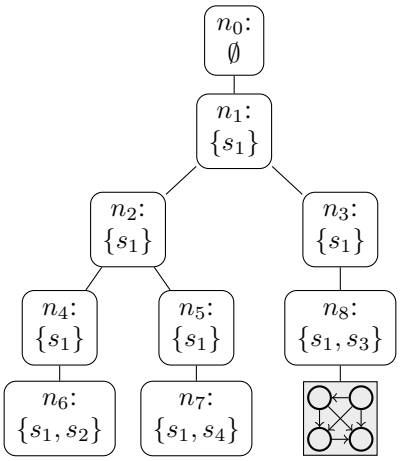
Theorem 3. *Reasoning and counting on AFs F in stable and complete semantics is fixed-parameter tractable when parameterized by* $\mathtt{btw}_{\mathcal{C}}(F)$ *for* $\mathcal{C} \in \{ACYC, NOEVEN\}$ *if a respective backdoor is given.*

We will show this using *nice* tree decompositions, where we restrict the different types of nodes: a tree-decomposition $(\mathcal{T}, \mathcal{X})$ is called *nice* if $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ is a rooted tree with an empty bag in the root node, and if each node $t \in \mathcal{T}$ (shorthand notion for $t \in V_{\mathcal{T}}$) is of one of the following types:

1. *Leaf*: t has no children in $\mathcal{T}$,
2. *Forget*: t has one child t', and $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$,
3. *Insert*: t has one child t', and $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$,
4. *Join*: t has two children t', t'', with $X_t = X_{t'} = X_{t''}$.

Moreover, we require each subset-maximal bag to (also) appear in the bag of a leaf of the TD. Any tree-decomposition can be transformed into a nice tree decomposition with the same width in time $O(|V_{\mathcal{T}}|)$ [27].

Example 1 ctd. An illustration of a nice tree decomposition for the AF F with an acyclic component "attached" to n_8 is given below.



We will show that our modifications of the DP algorithm are suitable for each of the semantics in question separately.

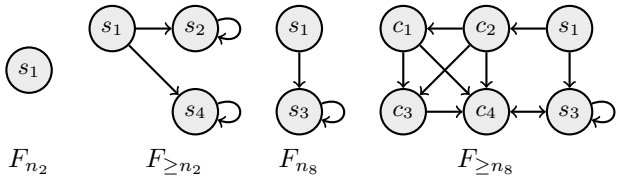
For each semantics, when carrying out the DP on an AF $F = (A, R)$, we will assume a suitable backdoor B to $\mathcal{C} \in \{ACYC, NOEVEN\}$ and a nice TD $(\mathcal{T}, \mathcal{X})$ of the torso G_F^B of width k is given.

We recall the modified notions introduced in Section 5 we use for TDs of torso graphs: For a set $B \subseteq A$, a tree decomposition $(\mathcal{T}, \mathcal{X})$ of torso graph G_F^B, and $t \in \mathcal{T}$, let $X_{\geq t}$ be the union of all bags $X_s \in \mathcal{X}$ and sets *components*(s) for every node s that occurs in the subtree of $\mathcal{T}$ rooted at t. Further, $X_{>t}$ shall denote $X_{\geq t} \setminus X_t$. Moreover, by $F_{\geq t}$ we denote the AF F projected to $X_{\geq t}$, i.e. $F_{\geq t} = (X_{\geq t}, R \cap (X_{\geq t} \times X_{\geq t}))$, and $F_t = (X_t, R \cap (X_t \times X_t))$.

Observe that for leaf nodes t of $\mathcal{T}$, we have $X_{>t} =$ *components*(t).

Example 1 ctd. To illustrate these notions, we consider the join node n_2 and the leaf node n_8. We have:

t	n_2	n_8
X_t	$\{s_1\}$	$\{s_1, s_3\}$
$X_{>t}$	$\{s_3, s_4\}$	$\{c_1, c_2, c_3, c_4\}$
$X_{\geq t}$	$\{s_1, s_3, s_4\}$	$\{s_1, s_3, c_1, c_2, c_3, c_4\}$
components(t)	(no leaf)	$\{c_1, c_2, c_3, c_4\}$



Stable Semantics

For stable semantics, we recall the notions of the algorithm presented by Charwat [8]. We will adapt the behavior of the algorithm in the leaf nodes. We start with the following auxiliary results. To this end, we utilize the notion of the *range* of a set of arguments S, denoted by $S^{\oplus}$. Let $F = (A, R)$ be an AF and let $S \subseteq A$ be a set of arguments, we define $S^{\oplus}_R = S \cup \{a \mid S \mapsto_R a\}$. Moreover, we use the *grounded extension* of an AF F [10], defined as the unique subset-minimal complete extension.

Lemma 1. *Let $F = (A, R)$ be an AF and let $B, S \subseteq A$ be sets of arguments. If S is $A \setminus B$-restricted stable in F, then $S' = S \setminus B$ is stable in $F' = (A', R')$, where $A' = A \setminus ((S \cap B)^{\oplus}_R \cup B)$ and $R' = R \cap (A' \times A')$.*

Proof. Towards contradiction, assume otherwise, i.e., S is $A \setminus B$-restricted stable in F, but there is an $x \in A'$ with $x \notin S'^{\oplus}_{R'}$. But then $x \notin B$, $x \notin (S \cap B)^{\oplus}_R$, and $x \notin (S \cap (A \setminus B))^{\oplus}_R$ in F, i.e., S is not $A \setminus B$-restricted stable in F, a contradiction. □

Lemma 2 (15)**.** *Let $F = (A, R)$ be an AF, and let $B \subseteq A$ be a $\mathcal{C}$-backdoor to $\mathcal{C} \in \{$ACYC, NOEVEN$\}$. Moreover, let $X \subseteq B$. There is a polynomial time algorithm that either returns the unique set S that is $(A \setminus B)$-restricted stable in F with $S \cap B = X$ or correctly concludes that no such set exists.*

Proof. We perform the following algorithm: First, we check if X is conflict-free in F, and stop if this is not the case. Instead of working on the set X, we will define the *labeling* $\lambda : B \to \{$*IN*, *DEF*$\}$ in the following way: $\lambda(x) =$ *IN* if $x \in X$, $\lambda(x) =$ *DEF* if $x \in B \setminus X$. From this, we calculate the (modified) *propagation* $\lambda^* : A \to \{$*IN*, *DEF*$\}$ as follows: we initialize $\lambda^*(x) = \lambda(x)$ for $x \in B$. Then, we apply the following two rules for each $x \in A \setminus B$:

1. Set $\lambda^*(x) =$ *IN* if all attackers y of x in F have $\lambda^*(y) =$ *DEF*.
2. Set $\lambda^*(x) =$ *DEF* if at least one attacker y of x in F has $\lambda^*(y) =$ *IN*.

After exhaustively applying these rules, we check if every argument in A has a value for λ^*. If this is the case, we return $S = \{x \mid \lambda^*(x) = \mathit{IN}\}$, otherwise we concludes that no such set exists. Clearly, all of these steps can be performed in polynomial time. By construction, this algorithm returns only $(A \setminus B)$-restricted stable sets. It remains to show that if an $(A \setminus B)$-restricted stable set $S \supset X$ exists for a given X, the algorithm finds it. Assume such a set S exists, let $\mu(x) : A \to \{\mathit{IN}, \mathit{DEF}\}$ be as follows: $\mu(x) = \mathit{IN}$ if $x \in S$, $\mu(x) = \mathit{DEF}$ if $x \in A \setminus S$. We show by induction on the number of applications of our propagation rule that $\mu(x) = \lambda^*(x)$ for all $x \in A$. We know by construction that μ and λ^* coincide on B. Now assume λ' is the labeling obtained after i applications of our rules and that x is the $i+1$-th argument that is labeled according to the rules. We distinguish two cases:

1. If x is added according to rule 1, we need to show that $\mu(x) = \mathit{IN}$. By induction hypothesis, $\lambda'(y) = \mu(y) = \mathit{DEF}$ for all attackers y of x in F. As S is $(A \setminus B)$-stable, it follows $\mu(x) = \mathit{IN}$.
2. If x is added according to rule 2, we need to show that $\mu(x) = \mathit{DEF}$. By induction hypothesis, $\lambda'(y) = \mu(y) = \mathit{IN}$ for some attacker y of x in F. Again, as S is $(A \setminus B)$-stable, it follows $\mu(x) = \mathit{DEF}$.

Finally, note that λ^* labels every argument in A in case such a set S exists: Effectively, the algorithm computes the grounded extension S' of the AF $F' = (A', R')$ with $A' = A \setminus (X_R^{\oplus} \cup B)$ and $R' = R \cap (A' \times A')$ (10). As F' contains no even length cycles, the grounded extensions is the only complete extension [12] and thus the only candidate for being a stable extension. By Lemma 1, the AF F' has a stable extension, hence, the calculated grounded extension is stable in F'. From this it follows that every argument in A is labeled by λ^*. □

In the following, we consider three colors *in*, *def*, *out*, meaning "in an extension", "defeated (attacked) by an extension", and "neither" (*out*). Note that these colors do not necessarily correspond to the labels used in the proof of Lemma 2. Formally, for a bag $t \in \mathcal{T}$, a *coloring* is a function

$$C_t : X_t \to \{\mathit{in}, \mathit{def}, \mathit{out}\}.$$

The extensions of a coloring C_t, denoted by $e_t(C_t)$, are defined as the set of $X_{>t}$-restricted stable sets S of $F_{\leq t}$, which satisfy the following conditions for each $a \in X_t$:

1. $C_t(a) = \mathit{in}$ iff $a \in S$,
2. $C_t(a) = \mathit{def}$ iff $S \mapsto a$, and
3. $C_t(a) = \mathit{out}$ iff $a \notin S$ and $S \not\mapsto a$.

A coloring C_t is a *valid coloring*, if $e_t(C_t) \neq \emptyset$. By $[C_t]$ we denote the set $\{a \mid C_t(a) = \mathit{in}\}$.

Colorings are an aggregated representation of (restricted) extensions, projected to the arguments that appear in a bag. If we know the valid colorings for each bag, we can reason and count efficiently on F. As we assume the bag at the root node to be empty, the valid colorings at the root node indeed capture the stable extensions of the whole AF F.

We want to establish that the DP algorithm yields the valid colorings at each node. We will do this for each node type separately, utilizing the auxiliary notion of *v*-

colorings, i.e. those colorings defined by the DP. Ultimately, we need to show that the v-colorings and valid colorings coincide at each node.

The DP for the node types forget, insert, and join, our DP will have the same behavior as the original DP as presented by Charwat [8]. For this, we recall the following notation. Let C, D be colorings:

$$(C - a)(b) = C(b) \text{ for each } b \in A \setminus \{a\}$$

$$(C + a)(b) = \begin{cases} C(b) \text{ if } b \in A \setminus \{a\} \\ \textit{def} \quad \text{if } b = a \text{ and } [C] \mapsto a \\ \textit{out} \quad \text{if } b = a \text{ and } [C] \not\mapsto a \end{cases}$$

$$(C \dot{+} a)(b) = \begin{cases} \textit{in} \quad \text{if } b = a \text{ or } C(b) = \textit{in} \\ \textit{def} \quad \text{if } a \neq b \text{ and } ((a,b) \in F_t \text{ or } C(b) = \textit{def}) \\ \textit{out} \quad \text{if } a \neq b, C(b) = \textit{out}, (a,b) \notin F_t \end{cases}$$

$$(C \bowtie D)(b) = \begin{cases} \textit{in} \quad \text{if } C(b) = D(b) = \textit{in} \\ \textit{def} \quad \text{if } C(b) = \textit{def} \text{ or } D(b) = \textit{def} \\ \textit{out} \quad \text{if } C(b) = D(b) = \textit{out} \end{cases}$$

Let $t \in \mathcal{T}$ be a *forget* node with a child t', and let a be the argument removed in X_t. If C is a v-coloring for t' and $C(a) \neq \textit{out}$, then $C - a$ is a v-coloring for t. If in t' v-colorings and valid colorings coincide, they then coincide in t [8, Lemma 3.23].

Let $t \in \mathcal{T}$ be an *insert* node with a child t', and let a be the argument inserted in X_t. If C is a v-coloring for t', then $C + a$ is a v-coloring for t. Moreover, if $(a,a) \notin R$, $[C] \not\mapsto a$, and $a \not\mapsto [C]$, then $C \dot{+} a$ is a v-coloring for t. If in t' v-colorings and valid colorings coincide, they then coincide in t [8, Lemma 3.21].

Let $t \in \mathcal{T}$ be a *join* node with children t' and t''. If C is a v-coloring for t' and D is a v-coloring for t'', and $[C] = [D]$, then $C \bowtie D$ is a v-coloring for t. If in t' and t'' v-colorings and valid colorings coincide, they then coincide in t [8, Lemma 3.25].

For *leaf* nodes, we deviate from the existing algorithm, and instead compute the following. Let $t \in \mathcal{T}$ be a *leaf* node. For each $X \subseteq X_t$, we apply Lemma 2 and get a set S with $S \cap X_t = X$, if it exists. Each such set S corresponds to the following v-coloring C_S:

- $C_S(a) = \textit{in}$ if $a \in X_t$ and $a \in S$,
- $C_S(a) = \textit{def}$ if $a \in X_t$ and $S \mapsto a$ in $F_{\geq t}$, and
- $C_S(a) = \textit{out}$ if $a \in X_t$ and neither $a \in S$ nor $S \mapsto a$ in $F_{\geq t}$.

By Lemma 2, the v-colorings yield only $X_{>t}$-restricted stable sets. Moreover, as we invoked Lemma 2 on every possible combination of the arguments X_t, the v-colorings contain all colorings corresponding to $X_{>t}$-restricted stable sets. Hence, v-colorings and valid colorings coincide. Note that the runtime of our algorithm for each leaf is $O(2^{|X_t|} \cdot |F_{\geq t}|^{O(1)})$.

By induction we conclude that all nodes characterize valid colorings. Reasoning and counting can be easily added to this algorithm, as discussed by Dvořák et al. [20].

Hence, the presented fixed-parameter tractable modification of the DP algorithm allows us to reason on F and count the stable extensions.

Complete Semantics

Similar to stable semantics, for complete semantics, we adapt algorithm presented by Charwat [8]. In particular, we will adapt the behavior of the algorithm in the leaf nodes.

We consider five colors *in*, *def*, *defp*, *out*, and *outp*, meaning "in an extension", "defeated (attacked) by an extension", provisionally defeated, neither *in* nor *def*, and provisionally neither *in* nor *def*, respectively. Formally, for a bag $t \in \mathcal{T}$, a *coloring* is a function

$$C_t : X_t \to \{\mathit{in}, \mathit{def}, \mathit{defp}, \mathit{out}, \mathit{outp}\}.$$

The extensions of a coloring C_t are defined via corresponding *labelings* [6]. A labeling λ on arguments A of an AF $F = (A, R)$ is a function $\lambda : A \to \{\mathit{IN}, \mathit{DEF}, \mathit{OUT}\}$. We write $\lambda_L = \{a \mid \lambda(a) = L\}$ for $L \in \{\mathit{IN}, \mathit{DEF}, \mathit{OUT}\}$. A labeling is conflict-free if $\lambda_{\mathit{IN}} \not\mapsto \lambda_{\mathit{IN}}$. A labeling is complete if for an argument $a \in A$:

1. $a \in \lambda_{\mathit{IN}}$ iff $\{b \mid (b, a) \in R\} \subseteq \lambda_{\mathit{DEF}}$
2. $a \in \lambda_{\mathit{DEF}}$ iff $\lambda_{\mathit{IN}} \mapsto a$
3. $a \in \lambda_{\mathit{OUT}}$ iff $\lambda_{\mathit{IN}} \not\mapsto a$ and $\lambda_{\mathit{OUT}} \mapsto a$

Caminada and Gabbay [6] showed that there is a one-to-one correspondence between complete extensions and complete labelings, i.e., for each $E \in \mathit{com}(F)$ there is one complete labeling λ with $E = \lambda_{\mathit{IN}}$.

Following Charwat [8], we will continue with the needed notions for the DP. A *B-restricted complete labeling* for F is a conflict-free labeling λ where the three above mentioned conditions for complete labelings only have to hold for all arguments $a \in B$.

For a node t of a TD $(\mathcal{T}, \mathcal{X})$ and coloring C_t, we denote by $l_t(C_t)$ the collection of all $X_{>t}$-restricted complete labelings λ for $F_{\geq t}$ which satisfy the following conditions for each $a \in X_t$:

- $C_t(a) = \mathit{in}$ iff $a \in \lambda_{\mathit{IN}}$
- $C_t(a) = \mathit{def}$ iff $a \in \lambda_{\mathit{DEF}}$ and $\lambda_{\mathit{IN}} \mapsto a$
- $C_t(a) = \mathit{defp}$ iff $a \in \lambda_{\mathit{DEF}}$ and $\lambda_{\mathit{IN}} \not\mapsto a$
- $C_t(a) = \mathit{out}$ iff $a \in \lambda_{\mathit{OUT}}$, $\lambda_{\mathit{IN}} \not\mapsto a$, $a \not\mapsto \lambda_{\mathit{IN}}$, and $\lambda_{\mathit{OUT}} \mapsto a$
- $C_t(a) = \mathit{outp}$ iff $a \in \lambda_{\mathit{OUT}}$, $\lambda_{\mathit{IN}} \not\mapsto a$, $a \not\mapsto \lambda_{\mathit{IN}}$, and $\lambda_{\mathit{OUT}} \not\mapsto a$

A coloring C_t is a *valid coloring*, if $l_t(C_t) \neq \emptyset$.

By $[C_t]$ we denote the set $\{a \mid C_t(a) = \mathit{in}\}$. Moreover, by $[C_t]_o$ we denote the set $\{a \mid C_t(a) = \mathit{out} \text{ or } C_t(a) = \mathit{outp}\}$ and by $[C_t]_d$ we denote the set $\{a \mid C_t(a) = \mathit{def} \text{ or } C_t(a) = \mathit{defp}\}$.

Again, colorings are an aggregated representation of (restricted) extensions, projected to the arguments that appear in a bag. If we know the valid colorings for each bag, we can reason and count efficiently on F. As we assume the bag at the root node to be empty, the valid colorings at the root node indeed capture the complete labelings (and, hence, extensions) of the whole AF F.

We want to establish that the DP algorithm yields the valid colorings at each node. We will do this for each node type separately, utilizing the auxiliary notion of *v-*

colorings, i.e. those colorings defined by the DP. Ultimately, we need to show that the v-colorings and valid colorings coincide at each node.

The DP for the node types forget, insert, and join, our DP will have the same behavior as the original DP as presented by Charwat [8]. For this, we recall the following notation. Let C, D be colorings:

$$(C - a)(b) = C(b) \text{ for each } b \in A \setminus \{a\}$$

$$(C + a)(b) = \begin{cases} C(b) \text{ if } b \in A \setminus \{a\} \\ \textit{def} \quad \text{if } b = a \text{ and } [C] \mapsto a \\ \textit{defp} \quad \text{if } b = a \text{ and } [C] \not\mapsto a \end{cases}$$

$$(C \dot{+} a)(b) = \begin{cases} C(b) \text{ if } b \in A \setminus \{a\} \text{ and} \\ \qquad C(b) \in \{\textit{in}, \textit{def}, \textit{defp}, \textit{out}\} \\ \textit{out} \quad \text{if } (b = a \text{ and } [C]_o \mapsto a) \text{ or} \\ \qquad (b \neq a \text{ and } a \mapsto b \text{ and } C(b) = \textit{outp}) \\ \textit{outp} \quad \text{otherwise} \end{cases}$$

$$(C \ddot{+} a)(b) = \begin{cases} \textit{in} \quad \text{if } b = a \\ C(b) \text{ if } (b \in A \setminus \{a\} \text{ and} \\ \qquad C(b) \in \{\textit{in}, \textit{def}, \textit{defp}, \textit{out}\} \\ \textit{def} \quad \text{if } b \neq a \text{ and } (a, b) \in F_t \text{ and} \\ \qquad C(b) = \textit{defp} \\ \textit{defp} \quad \text{if } b \neq a \text{ and } (a, b) \notin F_t \text{ and} \\ \qquad C(b) = \textit{defp} \end{cases}$$

$$(C \bowtie D)(b) = \begin{cases} \textit{in} \quad \text{if } C(b) = D(b) = \textit{in} \\ \textit{def} \quad \text{if } C(b) = \textit{def} \text{ or } D(b) = \textit{def} \\ \textit{defp} \quad \text{if } C(b) = D(b) = \textit{defp} \\ \textit{out} \quad \text{if } C(b) = \textit{out} \text{ or } D(b) = \textit{out} \\ \textit{outp} \quad \text{if } C(b) = D(b) = \textit{outp} \end{cases}$$

Let $t \in \mathcal{T}$ be a *forget* node with a child t', and let a be the argument removed in X_t. If C is a v-coloring for t' and $C(a) \notin \{\textit{defp}, \textit{outp}\}$, then $C - a$ is a v-coloring for t. If in t' v-colorings and valid colorings coincide, they then coincide in t [8, Lemma 3.41].

Let $t \in \mathcal{T}$ be an *insert* node with a child t', and let a be the argument inserted in X_t. If C is a v-coloring for t', then $C + a$ is a v-coloring for t. Moreover, if $[C] \not\mapsto a$ and $a \not\mapsto [C]$, then $C \dot{+} a$ is a v-coloring for t. Furthermore, if $(a, a) \notin R$, $[C] \not\mapsto a$, $a \not\mapsto [C]$, $[C]_o \not\mapsto a$, and $a \not\mapsto [C]_o$, then $C \ddot{+} a$ is a v-coloring for t. If in t' v-colorings and valid colorings coincide, they then coincide in t [8, Lemma 3.39].

Let $t \in \mathcal{T}$ be a *join* node with children t' and t''. If C is a v-coloring for t' and D is a v-coloring for t'', $[C] = [D]$, $[C]_o = [D]_o$, and $[C]_d = [D]_d$, then $C \bowtie D$ is a v-coloring for t. If in t' and t'' v-colorings and valid colorings coincide, they then coincide in t [8, Lemma 3.43].

For *leaf* nodes, we deviate from the existing algorithm, and instead compute the following. For each of the $O(3^k)$ labelings λ on X_t, we compute the propagation λ^* by initializing $\lambda^*(x) = \lambda(x)$ for $x \in X_t$, and then exhaustively applying the following rules [15] for the arguments $x \in X_{>t}$:

1. Set $\lambda^*(x) = \mathit{IN}$ if all attackers y of x in $F_{\geq t}$ have $\lambda^*(y) = \mathit{DEF}$.
2. Set $\lambda^*(x) = \mathit{DEF}$ if at least one attacker y of x in $F_{\geq t}$ has $\lambda^*(y) = \mathit{IN}$.
3. Set $\lambda^*(x) = \mathit{OUT}$ if all attackers y of x in $F_{\geq t}$ have either $\lambda^*(y) = \mathit{DEF}$ or $\lambda^*(y) = \mathit{OUT}$ and there is at least one attacker y of x in $F_{\geq t}$ with $\lambda^*(y) = \mathit{OUT}$.

It directly follows from [15] that the resulting labelings contain all X_t-restricted complete labelings. Finally, note that we can check in polynomial time whether a labeling is X_t-restricted complete. We obtain the set L of X_t-restricted complete labelings. We construct our *v-colorings* of X_t as follows: for each $\lambda \in L$, we get a coloring C_λ such that:

- $C_\lambda(a) = \mathit{in}$ if $a \in X_t$ and $a \in \lambda_{\mathit{IN}}$
- $C_\lambda(a) = \mathit{def}$ if $a \in X_t$, $a \in \lambda_{\mathit{DEF}}$ and $\lambda_{\mathit{IN}} \mapsto a$
- $C_\lambda(a) = \mathit{defp}$ if $a \in X_t$, $a \in \lambda_{\mathit{DEF}}$ and $\lambda_{\mathit{IN}} \not\mapsto a$
- $C_\lambda(a) = \mathit{out}$ if $a \in X_t$, $a \in \lambda_{\mathit{OUT}}$, $\lambda_{\mathit{IN}} \not\mapsto a$, $a \not\mapsto \lambda_{\mathit{IN}}$, and $\lambda_{\mathit{OUT}} \mapsto a$
- $C_\lambda(a) = \mathit{outp}$ if $a \in X_t$, $a \in \lambda_{\mathit{OUT}}$, $\lambda_{\mathit{IN}} \not\mapsto a$, $a \not\mapsto \lambda_{\mathit{IN}}$, and $\lambda_{\mathit{OUT}} \not\mapsto a$

By construction, *v-colorings* and valid colorings coincide.

Note that the runtime of our algorithm for each leaf is $O(3^k \cdot |F_{\geq t}|^{O(1)})$.

By induction we conclude that all nodes characterize valid colorings. Reasoning and counting can be easily added to this algorithm, as discussed by Dvořák et al. [20]. Hence, the presented fixed-parameter tractable modification of the DP algorithm allows us to reason on F and count the complete extensions.

C Details of the SAT Encoding for Computing the Optimal Backdoor-Treewidth

We define a propositional formula $\mathcal{E}(F, k)$. Given an AF $F = (A, R)$ and integer k, $\mathcal{E}(F, k)$ is satisfiable if and only if $\mathtt{btw}_{\mathit{ACYC}}(F) \leq k$. The encoding consists of three parts: first, we encode the acyclicity backdoor S; then, we encode the torso graph $G = (V, E)$; and finally, we use the elimination ordering to determine the torso graph's treewidth [31].

Elimination Orderings. The elimination ordering of a graph $G = (V, E)$, is a linear ordering $\prec$ of its vertices [5]. The width of the ordering, and the corresponding tree decomposition, is determined by the set E' of arcs. E' contains an arc (u, v) for each edge $\{u, v\} \in E$, such that $u \prec v$. Furthermore, E' contains *fill-in* edges: whenever $(u, v) \in E'$ and $(u, w) \in E'$, with $v \prec w$, then also $(v, w) \in E'$. The width is then the maximum out-degree of any vertex: $\max_{u \in V} |\{ (u, v) \in E' : v \in V \}|$.

Encoding. Table 2 shows the variables used in the SAT encoding. For brevity, we assume that arguments are ordered, i.e. whenever $u, v \in A$ such that $u \neq v$, then either

Var	Index Range	Semantics
b_u	$u \in A$	$u \in S$
$c_{u,v}$	$u \in A, v \in A$	There exists a path from u to v
$t_{u,v}$	$u \in A, v \in A, u \neq v$	u is adjacent to v's S-component
$o_{u,v}$	$u \in A, v \in A, u < v$	$u \prec v$
$a_{u,v}$	$u \in A, v \in A, u \neq v$	$(u, v) \in E'$

Table 2: Variables used in the SAT encoding.

$u < v$ or $u > v$. We use the shorthand $o^*_{u,v}$ to mean $o_{u,v}$ if $u < v$ or $\neg o_{v,u}$ otherwise. Furthermore we use $\{u, v\} \in R$ as a shorthand for $(u, v) \in R$ or $(v, u) \in R$.

We encode the acyclicity backdoor S as follows. First, we add for each argument $u \in A$ the acyclicity requirement $\neg c_{u,u}$. Next, we state that two arguments are connected if there is an arc between them and they are not part of the backdoor. We add for all $(u, v) \in R$ the clauses $\neg b_u \wedge \neg b_v \rightarrow c_{u,v}$. Further, we encode that connectedness is transitive using for each $u \in A$ and $(v, w) \in R$ the clauses $\neg b_w \wedge c_{u,v} \rightarrow c_{u,w}$. Finally, we add self-loops to S and add for all $(u, u) \in R$ the clauses b_u.

Although the edges of the torso graph are undirected, the arcs associated with the elimination ordering are directed from the lower ordered argument to the higher ordered argument. We establish that o is a linear ordering, by enforcing transitivity: for distinct $u, v, w \in A$, we add the clauses $o^*_{u,v} \wedge o*_{v,w} \rightarrow o^*_{u,w}$.

The first step towards encoding the torso graph is encoding t. First, we enforce for $t_{u,v}$ that $u \in S$ and $v \notin S$: we add for all $u, v \in A, u \neq v$ the clauses $t_{u,v} \rightarrow b_u$ and $t_{u,v} \rightarrow \neg b_v$. Next, we initialize t with all arguments $u, v \in A$, such that $\{u, v\} \in R$, and add the clauses $b_u \wedge \neg b_v \rightarrow t_{u,v}$. Finally, we propagate t: for all $u, v, w \in A$, such that $\{u, v\}, \{u, w\} \in R$ and $v \neq w$, we add the clauses $t_{u,w} \wedge \neg b_v \rightarrow t_{u,v}$.

We can now encode the edges of the torso graph in the form of the arc set E'. The adjacent arguments $u, v \in S$, are encoded by adding for all arguments $u, v \in A$, such that $\{u, v\} \in R$, the clauses $b_u \wedge b_v \wedge o^*_{u,v} \rightarrow a_{u,v}$. Further, we identify the remaining edges using t: for $u, v, w \in A$ such that $\{v, w\} \in R$ and $u \neq v$, we add the clauses $t_{u,w} \wedge b_v \wedge o^*_{u,v} \rightarrow a_{u,v}$.

The last step is adding the fill-in edges. We add for all distinct $u, v \in A$ the clauses $a_{u,v} \wedge a_{u,w} \wedge o^*_{v,w} \rightarrow a_{v,w}$.

Backdoor-Treewidth. In order to establish the backdoor-treewidth, we constrain for each $u \in A$ the cardinality, such that $|\{a_{u,v} : v \in A, u \neq v\}| \leq k$, using specific clauses called the totalizer cardinality constraints [3, 29]. We then, starting from the upper bound, $k = |V|$, encode $\mathcal{E}(F, k)$ and as long as the SAT solver returns satisfiable, we decrement k. The last k before the reach unsatisfiable is then $\texttt{btw}(F)$.

Symmetry Breaking. We can remove symmetries, by fixing the order of arguments outside of the S, as their ordering does not matter. We add for $u, v \in A, u < v$ the clauses $\neg b_u \wedge \neg b_v \rightarrow o^*_{u,v}$ and for $u, v \in A, u \neq v$ the clauses $\neg b_u \wedge b_v \rightarrow o^*_{u,v}$.

We used the encoding to compute the exact treewidth and backdoor-treewidth for the instances of scenario (B) (see Figure 8).

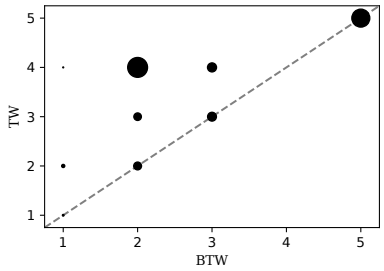
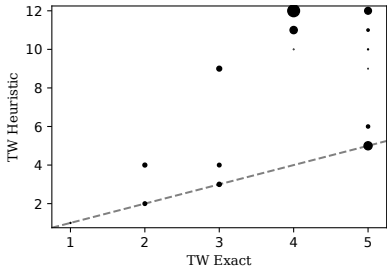


Figure 8: Comparison between exact backdoor-treewidth and treewidth. The size of the dot indicates the number of instances represented by the dot.

Comparison of Exact Widths vs. Heuristic Values

For the argBTW system we implemented a heuristic approach to obtain backdoor sets corresponding to torso graphs of small (but not necessarily minimal) width. In particular, we use an ASP encoding to approach the minimum *size* backdoor, which is then used to construct a torso. We found significant speedups with this method compared to the exact SAT encoding—in particular instances with larger sizes/widths turned out to be infeasible with the exact SAT encoding. Moreover, the treewidth of the resulting torso graphs was heuristically approached in the argBTW system.

Treewidth. In the following, we compare the treewidth upper bounds obtained by the argTW system (y-axis) with the exactly computed values by the SAT encoding (x-axis):

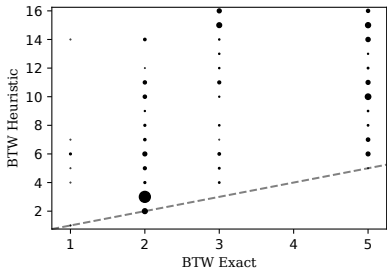


The size of the dot indicates the number of instances represented by the dot.

Moreover, we compare the mean value μ and standard deviation σ of the obtained values rounded to the last 2 decimal digits:

	μ	σ
TW Heuristic	9.07	3.55
TW Exact	4.04	0.87

Backdoor-Treewidth. In the following, we compare the backdoor-treewidth upper bounds obtained by the argTW system (y-axis) with the exactly computed values by the SAT encoding (x-axis):



The size of the dot indicates the number of instances represented by the dot.

Moreover, we compare the mean value μ and standard deviation σ of the obtained values rounded to the last 2 decimal digits:

	μ	σ
BTW Heuristic	7.99	4.88
BTW Exact	3.11	1.35