Backdoor Treewidth for SAT

Robert Ganian(✉), M.S. Ramanujan, and Stefan Szeider

Algorithms and Complexity Group, TU Wien, Vienna, Austria
{ganian,ramanujan,sz}@ac.tuwien.ac.at

Abstract. A strong backdoor in a CNF formula is a set of variables such that each possible instantiation of these variables moves the formula into a tractable class. The algorithmic problem of finding a strong backdoor has been the subject of intensive study, mostly within the parameterized complexity framework. Results to date focused primarily on backdoors of small size. In this paper we propose a new approach for algorithmically exploiting strong backdoors for SAT: instead of focusing on small backdoors, we focus on backdoors with certain structural properties. In particular, we consider backdoors that have a certain tree-like structure, formally captured by the notion of *backdoor treewidth*.

First, we provide a fixed-parameter algorithm for SAT parameterized by the backdoor treewidth w.r.t. the fundamental tractable classes Horn, Anti-Horn, and 2CNF. Second, we consider the more general setting where the backdoor decomposes the instance into components belonging to different tractable classes, albeit focusing on backdoors of treewidth 1 (i.e., acyclic backdoors). We give polynomial-time algorithms for SAT and #SAT for instances that admit such an acyclic backdoor.

1 Introduction

SAT is the problem of determining whether a propositional formula in conjuntive normal form (CNF) is satisfiable. Since SAT was identified as the first NP-complete problem, a significant amount of research has been devoted to the identification of "islands of tractability" or "tractable classes," which are sets of CNF formulas on which SAT is solvable in polynomial time. The notion of a strong backdoor, introduced by Williams et al. [28], allows one to extend these polynomial-time results to CNF formulas that do not belong to an island of tractability but are close to one. Namely, a *strong backdoor* is a set of variables of the given CNF formula, such that for all possible truth assignments to the variables in the set, applying the assignment moves the CNF formula into the island of tractability under consideration. In other words, using a strong backdoor consisting of k variables transforms the satisfiability decision for one general CNF formula into the satisfiability decision for 2^k tractable CNF formulas. A natural way of exploiting strong backdoors algorithmically is to search for *small* strong backdoors. For standard islands of tractability, such as the class of Horn

Supported by the Austrian Science Fund (FWF), project P26696. Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.

S. Gaspers and T. Walsh (Eds.): SAT 2017, LNCS 10491, pp. 20–37, 2017.
DOI: 10.1007/978-3-319-66263-3_2

formulas and the class of 2CNF formulas, one can find a strong backdoor of size k (if it exists) in time $f(k)L^c$ (where f is a computable function, c is a constant, and L denotes the length of the input formula) [22]; in other words, the detection of a strong backdoor of size k to Horn or 2CNF is *fixed-parameter tractable* [10]. The parameterized complexity of backdoor detection has been the subject of intensive study. We refer the reader to a survey article [16] for a comprehensive overview of this topic.

In this paper we propose a new approach for algorithmically exploiting strong backdoors for SAT. Instead of focusing on small backdoors, we focus on backdoors with certain structural properties. This includes backdoors of unbounded size and thus applies in cases that were not accessible by previous backdoor approaches. In particular, we consider backdoors that, roughly speaking, can be arbitrarily large but have a certain "tree-like" structure. Formally, this structure is captured in terms of the *treewidth* of a graph modeling the interactions between the backdoor and the remainder of the CNF formula (this is called the *backdoor treewidth* [14]). *Treewidth* itself is a well-established structural parameter that can be used to obtain fixed-parameter tractability of SAT [26]. The combination of strong backdoors and treewidth, as considered in this paper, gives rise to new tractability results for SAT, not achievable by backdoors or treewidth alone.

The notion of backdoor treewidth outlined above was recently introduced in the context of the *constraint satisfaction problem* (CSP) [14]. However, a direct translation of those results to SAT seems unlikely. In particular, while the results for CSP can be used "out-of-the-box" for CNF formulas of bounded clause width, additional machinery is required in order to handle CNF formulas of *unbounded* clause width.

The first main contribution of our paper is hence the following.

(1) SAT is fixed-parameter tractable when parameterized by the backdoor treewidth w.r.t. any of the following islands of tractability: Horn, Anti-Horn, and 2CNF (Theorem 1).

For our second main contribution, we consider a much wider range of islands of tractability, namely every island of tractability that is closed under partial assignments (a property shared by most islands of tractability studied in the literature). Moreover, we consider backdoors that split the input CNF formula into components where each of them may belong to a different island of tractability (we can therefore speak of an "archipelago of tractability" [15]). This is a very general setting, and finding such a backdoor of small treewidth is a challenging algorithmic task. It is not at all clear how to handle even the special case of backdoor treewidth 1, i.e., acyclic backdoors.

In this work, we take the first steps in this direction and settle this special case of acyclic backdoors. We show that if a given CNF formula has an acyclic backdoor into an archipelago of tractability, we can test its satisfiability in polynomial time. We also obtain an equivalent result for the model counting problem #SAT (which asks for the number of satisfying assignments of the given formula).

(2) SAT and #SAT are solvable in polynomial time for CNF formulas with an acyclic backdoor into any archipelago of tractability whose islands are closed under partial assignments (Theorems 2 and 3).

We note that the machinery developed for backdoor treewidth in the CSP setting cannot be used in conjunction with islands of tractability in the general context outlined above; in fact, we leave open even the existence of a polynomial-time algorithm for backdoor treewidth 2. An interesting feature of the algorithms establishing Theorems 2 and 3 is that they do not explicitly *detect* an acyclic backdoor. Instead, we only require the *existence* of such a backdoor in order to guarantee that our algorithm is correct on such inputs. In this respect, our results add to the rather small set of backdoor based algorithms for SAT (see also [13]) which only rely on the existence of a specific kind of backdoor rather than computing it, in order to solve the instance.

We now briefly mention the techniques used to establish our results. The general idea behind the proof of Theorem 1 is the translation of the given CNF formula F into a "backdoor-equivalent" CNF formula F' which has bounded clause width. Following this, we can safely perform a direct translation of F' into a CSP instance $\mathcal{I}$ which satisfies all the conditions for invoking the previous results on CSP [14]. For Theorems 2 and 3, we consider the biconnected components of the incidence graph of the CNF formula and prove that the existence of an acyclic backdoor imposes useful structure on them. We then design a dynamic programming algorithm which runs on the block decomposition of the incidence graph of the CNF formula and show that it correctly determines in polynomial time whether the input CNF formula is satisfiable (or counts the number of satisfying assignments, respectively) *as long as* there is an acyclic backdoor of the required kind.

2 Preliminaries

Parameterized Complexity. We begin with a brief review of the most important concepts of parameterized complexity. For an in-depth treatment of the subject we refer the reader to textbooks [8,10].

The instances of a parameterized problem can be considered as pairs (I, k) where I is the *main part* of the instance and k is the *parameter* of the instance; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* (FPT) if instances (I, k) of size n (with respect to some reasonable encoding) can be solved in time $O(f(k)n^c)$ where f is a computable function and c is a constant independent of k. The function f is called the *parameter dependence*.

We say that parameter X *dominates* parameter Y if there exists a computable function f such that for each CNF formula F we have $X(F) \leq f(Y(F))$ [25]. In particular, if X dominates Y and SAT is FPT parameterized by X, then SAT is FPT parameterized by Y [25]. We say that two parameters are *incomparable* if neither dominates the other.

Satisfiability. We consider propositional formulas in conjunctive normal form (CNF), represented as sets of clauses. That is, a *literal* is a (propositional) variable x or a negated variable $\overline{x}$; a *clause* is a finite set of literals not containing a complementary pair x and $\overline{x}$; a *formula* is a finite set of clauses. For a literal $l = \overline{x}$ we write $\overline{l} = x$; for a clause c we set $\overline{c} = \{\overline{l} \mid l \in c\}$. For a clause c, $var(c)$ denotes the set of variables x with $x \in c$ or $\overline{x} \in c$, and the *clause width* of c is $|var(c)|$. Similarly, for a CNF formula F we write $var(F) = \bigcup_{c \in F} var(c)$. The *length* (or *size*) of a CNF formula F is defined as $\sum_{c \in F} |c|$. We will sometimes use the a graph representation of a CNF formula F called the *incidence graph* of F and denoted $\mathsf{Inc}(F)$. The vertices of $\mathsf{Inc}(F)$ are variables and clauses of F and two vertices a, b are adjacent if and only if a is a clause and $b \in var(a)$.

A *truth assignment* (or *assignment*, for short) is a mapping $\tau : X \to \{0, 1\}$ defined on some set X of variables. We extend τ to literals by setting $\tau(\overline{x}) = 1 - \tau(x)$ for $x \in X$. $F[\tau]$ denotes the CNF formula obtained from F by removing all clauses that contain a literal x with $\tau(x) = 1$ and by removing from the remaining clauses all literals y with $\tau(y) = 0$; $F[\tau]$ is the *restriction* of F to τ. Note that $var(F[\tau]) \cap X = \emptyset$ holds for every assignment $\tau : X \to \{0, 1\}$ and every CNF formula F. An assignment $\tau : X \to \{0, 1\}$ *satisfies* a CNF formula F if $F[\tau] = \emptyset$, and a CNF formula F is satisfiable if there exists an assignment which satisfies F. In the SAT problem, we are given a CNF formula F and the task is to determine whether F is satisfiable.

Let $X \subseteq var(F)$. Two clauses c, c' are *X-adjacent* if $(var(c) \cap var(c')) \setminus X \neq \emptyset$. We say that two clauses c, d are *X-connected* if there exists a sequence $c = c_1, \ldots, c_r = d$ such that each consecutive c_i, c_{i+1} are X-adjacent. An *X-component* Z of a CNF formula F is then an inclusion-maximal subset of X-connected clauses, and its boundary is the set $var(Z) \cap X$. An $\emptyset$-component of a CNF formula F is also called a *connected component* of F.

A class $\mathcal{F}$ of CNF formulas is *closed under partial assignments* if, for each $F \in \mathcal{F}$ and each assignment τ of a subset of $var(F)$, it holds that $F[\tau] \in \mathcal{F}$. Examples of classes that are closed under partial assignment include 2CNF, Q-Horn, hitting CNF formulas and acyclic CNF formulas (see, e.g., the Handbook of Satisfiability [2]).

Backdoors and Tractable Classes for SAT. Backdoors are defined relative to some fixed class $\mathcal{C}$ of instances of the problem under consideration (i.e., SAT); such a class $\mathcal{C}$ is then often called the *base class*. One usually assumes that the problem is tractable for instances from $\mathcal{C}$, as well as that the recognition of $\mathcal{C}$ is tractable; here, tractable means solvable by a polynomial-time algorithm.

In the context of SAT, we define a *strong backdoor set* into $\mathcal{F}$ of a CNF formula F to be a set B of variables such that $F[\tau] \in \mathcal{F}$ for each assignment $\tau : B \to \{0, 1\}$. If we know a strong backdoor of F into $\mathcal{F}$, we can decide the satisfiability of F by looping over all assignments τ of the backdoor variables and checking the satisfiability of the resulting CNF formulas $F[\tau]$ (which belong to $\mathcal{F}$). Thus SAT decision is fixed-parameter tractable in the size k of the strong backdoor, assuming we are given such a backdoor as part of the input and $\mathcal{F}$ is tractable. We note that every natural base class $\mathcal{F}$ has the property that if

$B \subseteq var(F)$ is a backdoor of F into $\mathcal{F}$, then B is also a backdoor of every B-component of F into $\mathcal{F}$; indeed, each such B-component can be treated separately for individual assignments of B. We will hence assume that all our base classes have this property.

Here we will be concerned with three of the arguably most prominent polynomially tractable classes of CNF formulas: *Horn*, *Anti-Horn* and *2CNF*, defined in terms of syntactical properties of clauses. A clause is (a) *Horn* if it contains at most one positive literal, (b) *Anti-Horn* if it contains at most one negative literal, (c) *2CNF* if it contains at most two literals,[1]

A CNF formula belongs to the class *Horn*, *Anti-Horn*, or *2CNF* if it contains only Horn, Anti-Horn, or 2CNF clauses, respectively; each of these classes is known to be tractable and closed under partial assignments. It is known that backdoor detection for each of the classes listed above is FPT, which together with the tractability of these classes yields the following.

Proposition 1 ([16]). *SAT is fixed-parameter tractable when parameterized by the size of a minimum backdoor into $\mathcal{F}$, for each $\mathcal{F} \in \{$Horn, Anti-Horn, 2CNF$\}$.*

We note that in the literature also other types of backdoors (weak backdoors and deletion backdoors) have been considered; we refer to a survey article for examples [16]. In the sequel our focus lies on strong backdoors, and for the sake of brevity will refer to them merely as *backdoors*.

The Constraint Satisfaction Problem. Let $\mathcal{V}$ be a set of variables and $\mathcal{D}$ a finite set of values. A *constraint of arity ρ over $\mathcal{D}$* is a pair (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 $var(C) = \{x_1, \ldots, x_\rho\}$ is called the *scope* of C. An *assignment* $\alpha : X \to \mathcal{D}$ is a mapping of a set $X \subseteq \mathcal{V}$ of variables. An assignment $\alpha : X \to \mathcal{D}$ *satisfies* a constraint $C = ((x_1, \ldots, x_\rho), R)$ if $var(C) \subseteq X$ and $(\alpha(x_1), \ldots, \alpha(x_\rho)) \in R$. For a set $\mathcal{I}$ of constraints we write $var(\mathcal{I}) = \bigcup_{C \in \mathcal{I}} var(C)$ and $\text{rel}(\mathcal{I}) = \{R \mid (S, R) \in C, C \in \mathcal{I}\}$. A finite set $\mathcal{I}$ of constraints is *satisfiable* if there exists an assignment that simultaneously satisfies all the constraints in $\mathcal{I}$. The *Constraint Satisfaction Problem* (CSP, for short) asks, given a finite set $\mathcal{I}$ of constraints, whether $\mathcal{I}$ is satisfiable.

Next, we will describe how a partial assignment alters a given CSP instance. 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 $R \in \mathcal{D}^\rho$, we denote by $C|_\alpha$ the constraint (S', R') obtained from C as follows. R' is obtained 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 $x_i \in X$. S' is obtained from S by deleting all variables x_i with $x_i \in X$. For a set $\mathcal{I}$ of constraints we define $\mathcal{I}|_\alpha$ as $\{C|_\alpha \mid C \in \mathcal{I}\}$. It is important to note that there is a crucial distinction between assignments in SAT and assignments in CSP: while in SAT one deletes all clauses which are already satisfied by a given assignment,

[1] A clause containing exactly two literals is also known as a *Krom clause*.

in CSP this is not the case – instead, constraints are restricted to the tuples which match the assignment (but never deleted).

A *constraint language* (or *language*, for short) Γ over a domain $\mathcal{D}$ is a set of relations (of possibly various arities) over $\mathcal{D}$. By $\mathrm{CSP}(\Gamma)$ we denote CSP restricted to instances $\mathcal{I}$ with $\mathrm{rel}(\mathcal{I}) \subseteq \Gamma$. A constraint language is *tractable* if for every finite subset $\Gamma' \subseteq \Gamma$, the problem $\mathrm{CSP}(\Gamma')$ can be solved in polynomial time. Let Γ be a constraint language and $\mathcal{I}$ be an instance of CSP. A variable set X is a (*strong*) *backdoor* to $\mathrm{CSP}(\Gamma)$ if for each assignment $\alpha : X \to \mathcal{D}$ it holds that $\mathcal{I}|_\alpha \in \mathrm{CSP}(\Gamma)$. A language is *closed under partial assignments* if for each $\mathcal{I} \in \mathrm{CSP}(\Gamma)$ and for each assignment α, it holds that $\mathcal{I}|_\alpha \in \mathrm{CSP}(\Gamma)$. Finally, observe that the notions of being *X-adjacent*, *X-connected* and being a *X-component* can be straightforwardly translated to CSP. For example, a CSP instance containing three constraints $((a, b, d), R_1)$, $((c, d), R_2)$, and $((e, b), R_3)$ would contain two $\{b\}$-components: one containing $((a, b, d), R_1)$, $((c, d), R_2)$ and the other containing $((e, b), R_3)$.

Each SAT instance (i.e., CNF formula) admits a direct encoding into a CSP instance (over the same variable set and with domain $\{0, 1\}$), which transforms each clause into a relation containing all tuples which do not invalidate that clause. Note that this will exponentially increase the size of the instance if the CNF formula contains clauses of unbounded clause width; however, for any fixed bound on the clause width of the original CNF formula, the direct encoding into CSP will only increase the bit size of the instance by a constant factor.

Treewidth and Block Decompositions. The set of *internal vertices* of a path P in a graph is denoted by $V_{\mathsf{int}}(P)$ and is defined as the set of vertices in P which are not the endpoints of P. We say that two paths P_1 and P_2 in an undirected graph are *internally vertex-disjoint* if $V_{\mathsf{int}}(P_1) \cap V_{\mathsf{int}}(P_2) = \emptyset$. Note that under this definition, a path consisting of a single vertex is internally vertex-disjoint to every other path in the graph.

The graph parameter *treewidth* will be of particular interest in the context of this paper. Let G be a simple, undirected, finite graph with vertex set $V = V(G)$ and edge set $E = E(G)$. A *tree decomposition* of G is a pair $(\{B_i : i \in I\}, T)$ where $B_i \subseteq V$, $i \in I$, and T is a tree with elements of I as nodes such that (a) for each edge $uv \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$, and (b) for each vertex $v \in V$, $T[\{i \in I \mid v \in B_i\}]$ is a (connected) tree with at least one node. The *width* of a tree decomposition is $\max_{i \in I} |B_i| - 1$. The *treewidth* [21,24] of G is the minimum width taken over all tree decompositions of G and it is denoted by $\mathsf{tw}(G)$. We call the elements of I *nodes* and B_i *bags*. It is well known that, for every clique over $Z \subseteq V(G)$ in G, it holds that every tree decomposition of G contains an element B_i such that $Z \subseteq B_i$ [21].

We now recall the definitions of *blocks* and *block decompositions* in a graph. A *cut-vertex* in an undirected graph H is a vertex whose removal disconnects the connected component the vertex belongs to.

A maximal connected subgraph without a cut-vertex is called a *block*. Every block of a graph G is either a maximal 2-connected subgraph, or an isolated vertex or a path on 2 vertices (see, e.g., [9]).

By maximality, different blocks of a graph H overlap in at most one vertex, which is then easily seen to be a cut-vertex of H. Therefore, every edge of H lies in a unique block and H is the union of its blocks.

Let A denote the set of cut-vertices of H and B the set of its blocks. The *block-graph* of H is the bipartite graph on $A \cup B$ where $a \in A$ and $b \in B$ are adjacent if and only if $a \in b$. The set of vertices in B are called *block-vertices*.

Proposition 2 ([9]). *The block-graph of a connected undirected graph is a tree.*

Due to the above proposition, we will henceforth refer to block-graphs of connected graphs as block-trees. Furthermore, this proposition implies that the block-graph of a disconnected graph is precisely the disjoint union of the block-trees of its connected components. As a result, we refer to block-graphs in general as block-forests. Finally, it is straightforward to see that the leaves of the block-tree are all block-vertices.

The block decomposition of a connected graph G is a pair $(T, \eta : V(T) \rightarrow 2^{V(G)})$ where T is the block-tree, and (a) for every $t \in V(T)$ such that t is a block-vertex, $G[\eta(t)]$ is the block of G corresponding to this block-vertex and (b) for every $t \in V(T)$ such that t is a cut-vertex, $\eta(t) = \{t\}$. For a fixed *root* of the tree T and a vertex $t \in V(T)$, we denote by $\mathsf{child}(t)$ the set of children of t with respect to this root vertex.

Proposition 3 ([7,18]). *There is an algorithm that, given a graph G, runs in time $\mathcal{O}(m+n)$ and outputs the block decomposition of G.*

3 Backdoor Treewidth

The core idea of *backdoor treewidth* is to fundamentally alter how the *quality* of a backdoor is measured. Traditionally, the aim has always been to seek for backdoors of small *size*, since these can easily be used to obtain fixed-parameter algorithms for SAT. Instead, one can define the treewidth of a backdoor—obtained by measuring the treewidth of a graph obtained by "collapsing" the CNF formula into the backdoor—and show that this is more beneficial than focusing merely on its size. In line with existing literature in graph algorithms and theory, we call the resulting "collapsed graph" the *torso*.

Definition 1. *Let F be a CNF formula and X be a backdoor in F to a class $\mathcal{F}$. Then the* X-torso graph G_F^X *of F is the graph whose vertex set is X and where two variables x, y are adjacent if and only if there exists an X-component A such that $x, y \in var(A)$. The treewidth of X is then the treewidth of G_F^X, and the* backdoor treewidth *of F w.r.t. $\mathcal{F}$ is the minimum treewidth of a backdoor into $\mathcal{F}$.*

Given the above definition, it is not difficult to see that backdoor treewidth w.r.t. $\mathcal{F}$ is upper-bounded by the size of a minimum backdoor to $\mathcal{F}$, but can be arbitrarily smaller than the latter. As a simple example of this behavior, consider the CNF formula $\{\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{i-1}, x_i\}\}$, which does not contain a

constant-size backdoor to Horn but has backdoor treewidth 1 w.r.t. Horn (one can use, e.g., a backdoor containing every variable with an even index). This motivates the design of algorithms for SAT which are fixed-parameter tractable when parameterized not by the *size* of a smallest backdoor into a particular base class but by the value of the backdoor treewidth with respect to the base class. Our first main contribution is the following theorem.

Theorem 1. *SAT is fixed-parameter tractable when parameterized by the backdoor treewidth w.r.t. any of the following classes: Horn, Anti-Horn, 2CNF.*

In order to prove Theorem 1, we first show that a backdoor of small treewidth can be used to design a fixed-parameter algorithm for solving SAT *if* such a backdoor is provided in the input; we note that the proof of this claim proceeds along the same lines as the proof for the analogous lemma in the case of constraint satisfaction problems [14].

Lemma 1. *Let F be a CNF formula and let X be a strong backdoor of F into a tractable class $\mathcal{F}$. There is an algorithm that, given F and X, runs in time $2^{tw(G_F^X)}|F|^{\mathcal{O}(1)}$ and correctly decides whether F is satisfiable or not.*

Proof. We prove the lemma by designing an algorithm which constructs an equivalent CSP instance $\mathcal{I}$ (over domain $\{0,1\}$) and then solves the instance in the specified time bound. The variables of $\mathcal{I}$ are precisely the set X. For each X-component Z of F with boundary B, we add a constraint c_B into $\mathcal{I}$ over $var(Z)$, where c_B contains one tuple for each assignment of B which can be extended to a satisfying assignment of Z. For instance, if $B = \{a, b, c\}$ and the only assignment which can be extended to a satisfying assignment for Z is $a \mapsto 0, b \mapsto 1, c \mapsto 1$, then c_B will have the scope (a, b, c) and the relation with a single tuple $(0, 1, 1)$.

We note that since B is a backdoor of Z to $\mathcal{F}$ of size at most $k = G_F^X$ (as follows from the fact that B is a clique in G_F^X and hence must fully lie in some bag of T, and from our discussion following the introduction of backdoors), we can loop through all assignments of B and test whether each such assignment can be extended to satisfy Z or not in time at most $2^k \cdot |B|^{\mathcal{O}(1)}$. Consequently, we can construct $\mathcal{I}$ from F and X in time $2^k|F|^{\mathcal{O}(1)}$. As an immediate consequence of this construction, we see that any assignment satisfying $\mathcal{I}$ can be extended to a satisfying assignment for F, and vice-versa the restriction of any satisfying assignment for F onto X is a satisfying assignment in $\mathcal{I}$.

Next, in order to solve $\mathcal{I}$, we recall the notion of the *primal graph* of a CSP instance. The primal graph of $\mathcal{I}$ is the graph whose vertex set is X and where two vertices a, b are adjacent iff there exists a constraint whose scope contains both a and b. Observe that in the construction of $\mathcal{I}$, two variables a, b will be adjacent if and only if they occur in some X-component of F, i.e., a, b are adjacent in $\mathcal{I}$ iff they are adjacent in G_F^X. Hence the primal graph of $\mathcal{I}$ is isomorphic to G_F^X, and in particular the primal graph of $\mathcal{I}$ must have treewidth at most k. To conclude the proof, we use the well-known fact that boolean CSP can be solved in time $2^t \cdot n^{\mathcal{O}(1)}$, where n is the number of variables and t the treewidth of the primal graph [27]. □

With Lemma 1 in hand, it remains to show that a backdoor of small treewidth can be found efficiently (if such a backdoor exists). The main tool we will use in this respect is the following result, which solves the problem of finding backdoors of small treewidth in the context of CSP for classes defined via finite languages; we note that backdoor treewidth on backdoors for CSP is defined analogously as for SAT.

Proposition 4 ([14]). *Let Γ be a finite language. There exists a fixed-parameter algorithm which takes as input a CSP instance $\mathcal{I}$ and a parameter k, and either finds a backdoor of treewidth at most k into CSP(Γ) or correctly determines that no such backdoor exists.*

In order to prove Proposition 4, Ganian et al. [14] introduced a subroutine that is capable of replacing large parts of the input CSP by a strictly smaller CSP in such a way that the existence of a backdoor of treewidth at most k into CSP(Γ) is not affected. Their approach was inspired by the graph replacement tools dating back to the results of Fellows and Langston [11] and further developed by Arnborg, Bodlaender, and other authors [1,3–5,12]. Subsequently, they utilized the *recursive-understanding* technique, introduced by Grohe et al. [17] to show that as long as the instance has a size exceeding some function of k, then it is possible to find a large enough part of the input instance which can then be strictly reduced. Their theorem then follows by a repeated application of this subroutine, followed by a brute-force step at the end when the instance has size bounded by a function of k.

There are a few obstacles which prevent us from directly applying Proposition 4 to our SAT instances and backdoors to our classes of interest (Horn, Anti-Horn and 2CNF). First of all, while SAT instances admit a direct encoding into CSP, in case of unbounded clause width this can lead to an exponential blowup in the size of the instance. Second, the languages corresponding to Horn, Anti-Horn and 2CNF in such a direct encoding are not finite. Hence, instead of immediately using the direct encoding, we will proceed as follows: given a CNF formula F, we will first construct an auxiliary CNF formula F' of clause width at most 3 which is equivalent as far as the existence of backdoors is concerned. We note that the CNF formula F' constructed in this way will *not* be satisfiability-equivalent to F. But since F' has bounded clause width, we can then use a direct encoding of F' into CSP and apply Proposition 4 while circumventing the above obstacles.

Lemma 2. *Let $\mathcal{F} \in \{$Horn, Anti-Horn, 2CNF$\}$. There exists a polynomial-time algorithm which takes as input a CNF formula F and outputs a 3-CNF formula F' such that $var(F) = var(F')$ with the following property: for each $X \subseteq var(F)$, X is a backdoor of F into $\mathcal{F}$ if and only if X is a backdoor of F' into $\mathcal{F}$.*

Proof. We first describe the construction and then complete the proof by arguing that the desired property holds. We construct the CNF formula F' as follows: for each clause $c \in F$ of width at least 4, we loop over all sets of size 3 of literals from c and add each such set into F; afterwards, we remove c. Observe that

$|F'| \leq (2 \cdot |var(F)|)^3$ and each clause in F' is either equal to or "originated from" at least one clause in F.

Now consider a set $X \subseteq var(F)$ which is a backdoor of F into $\mathcal{F}$ (for an arbitrary choice of $\mathcal{F} \in \mathcal{H}$). We claim that X must also be a backdoor of F' into $\mathcal{F}$. Indeed, if $\mathcal{F} =$ Horn then for each assignment τ of X, each clause c in $F[\tau]$ must be a horn clause, i.e., cannot contain more than one positive literal. But then each c' in $F'[\tau]$ that originated from taking a subset of literals from c must also be a horn clause. If $\mathcal{F} =$ 2CNF then for each assignment τ of X, each clause c in $F[\tau]$ must contain at most two literals; once again, each c' in $F'[\tau]$ that originated from taking a subset of literals from c must also contain at most 2 literals. Finally, the same argument shows that the claim also holds if $\mathcal{F} =$ Anti-Horn.

On the other hand, consider a set $X \subseteq var(F)$ which is a backdoor of F' into some $\mathcal{F} \in \mathcal{H}$. Once again, we claim that X must also be a backdoor of F into $\mathcal{F}$. Indeed, let $\mathcal{F} =$ Horn and assume for a contradiction that there exists an assignment τ of X such that $F[\tau]$ contains a clause c with at least two positive literals, say a and b. Then $F'[\tau]$ must either also contain c, or it must contain a subset c' of c such that $a, b \in c'$; in either case, we arrive at a contradiction with X being a backdoor of F' into Horn. The argument for $\mathcal{F} =$ Anti-Horn is, naturally, fully symmetric. Finally, let $\mathcal{F} =$ 2CNF and assume for a contradiction that there exists an assignment τ of X such that $F[\tau]$ contains a clause c or width at least 3; let us pick three arbitrary literals in c, say a_1, a_2, a_3, and observe that these cannot be contained in X. Then, by construction, $F'[\tau]$ contains the clause $\{a_1, a_2, a_3\}$—contradicting the fact that X is a backdoor of F' into 2CNF. □

Our final task in this section is to use Lemma 2 to obtain an algorithm to detect a backdoor of small treewidth, which along with Lemma 1 implies Theorem 1.

Lemma 3. *Let $\mathcal{F} \in \{$Horn, Anti-Horn, 2CNF$\}$. There exists a fixed-parameter algorithm which takes as input a CNF formula F and a parameter k, and either finds a backdoor of treewidth at most k into $\mathcal{F}$ or correctly determines that no such backdoor exists.*

Proof. We begin by invoking Lemma 2 to obtain a 3-CNF formula F' which preserves the existence of backdoors. Next, we construct the direct encoding of F' into a CSP instance $\mathcal{I}$; since F' has bounded clause width, it holds that $|\mathcal{I}| \in \mathcal{O}(|F'|)$. For each choice of class $\mathcal{F}$, we construct the language Γ corresponding to the class in the setting of CSPs with arity at most 3. Specifically, if $\mathcal{F}$ is 2CNF, then Γ will contain all relations of arity at most 2; if $\mathcal{F}$ is Horn, then for each Horn clause of width at most 3, Γ will contain a relation with all tuples that satisfy the clause; and analogously for Anti-Horn. Finally, since Γ is a finite language, we invoke Proposition 4 to compute a backdoor of width at most k or correctly determine that no such backdoor exists in $\mathcal{I}$. Correctness follows from Lemma 2, the equivalence of $\mathcal{F}$ and CSP(Γ), and the natural correspondence of backdoors of F' into $\mathcal{F}$ to backdoors of $\mathcal{I}$ into CSP(Γ). □

4 Acyclic Backdoors to Scattered Classes for SAT

In this section, we build upon the notion of so-called *scattered classes* [15] and backdoor treewidth to introduce a new polynomial time tractable class of CNF formulas.

Definition 2. *Let $\mathcal{U} = \{\mathcal{F}_1, \ldots, \mathcal{F}_r\}$ be a set of classes of CNF formulas. For a CNF formula F, we say that $X \subseteq var(F)$ is a backdoor of F into $\mathcal{U}$ if for every X-component F' and every partial assignment τ to $var(F') \cap X$, the CNF formula $F'[\tau]$ belongs to a class in $\mathcal{U}$.*

We let $\mathsf{btw}_{\mathcal{U}}(F) = \min\{\mathsf{tw}(G_F^X) \mid X \text{ is a strong backdoor of } F \text{ into } \mathcal{U}\}$, and observe that if $\mathcal{U} = \{\mathcal{F}\}$ then backdoors into $\mathcal{U}$ coincide with backdoors into $\mathcal{F}$. Next, we define the general property we will require for our base classes. We note that these precisely coincide with the notion of *permissive classes* [23], and that being *permissively tractable* (see Definition 3) is clearly a necessary condition for being able to use any sort of backdoor into $\mathcal{F}$.

Definition 3. *Let $\mathcal{F}$ be a class of CNF formulas closed under partial assignments. Then $\mathcal{F}$ is called* permissively tractable *if there is a polynomial time algorithm that, given a CNF formula F, either correctly concludes that $F \notin \mathcal{F}$ or correctly decides whether F is satisfiable.*

Similarly, the class $\mathcal{F}$ is called #-permissively tractable *if there is a polynomial time algorithm that, given a CNF formula F either correctly concludes that $F \notin \mathcal{F}$ or correctly returns the number of satisfying assignments of F.*

We now state the two main results of this section.

Theorem 2. *Let $\mathcal{U} = \{\mathcal{F}_1, \ldots, \mathcal{F}_r\}$ be a set of permissively tractable classes. There is a polynomial time algorithm that decides whether any given CNF formula F with $\mathsf{btw}_{\mathcal{U}}(F) = 1$ is satisfiable.*

Theorem 3. *Let $\mathcal{U} = \{\mathcal{F}_1, \ldots, \mathcal{F}_r\}$ be a set of #-permissively tractable classes. There is a polynomial time algorithm that counts the number of satisfying assignments for any given CNF formula F with $\mathsf{btw}_{\mathcal{U}}(F) = 1$.*

We call S an *acyclic* (strong) backdoor of F into $\mathcal{U}$ if it is a backdoor of F into $\mathcal{U}$ and the treewidth of the S-torso graph G_F^S is 1. Let us now illustrate a general high-level example of CNF formulas that can now be considered as polynomial time tractable due to Theorem 2. Let $\mathcal{U}$ contain the tractable class *Q-Horn* [6] and the tractable class of *hitting CNF formulas* [19,20]. The class *Q-Horn* is a proper superset of 2CNF, Horn and Anti-Horn, and *hitting CNF formulas* are those CNF formulas where, for each pair of clauses, there exists a variable x which occurs positively in one clause and negatively in the other. We can solve any CNF formula F which is iteratively built from "building blocks", each containing at most 2 backdoor variables such that any assignment of these

variables turns the block into a hitting CNF formula or Q-Horn formula. Next we provide one example of such a building block (with backdoor variables x_1, x_2).

$$F' = \big\{\{x_1, x_2, a, b, c\}, \{\overline{x_1}, x_2, \overline{a}, \overline{c}\}, \{\overline{x_1}, x_2, a, c\}, \{x_1, \overline{x_2}, \overline{a}, \overline{b}, c\}, \\ \{\overline{x_1}, \overline{x_2}, a, b, c\}, \{\overline{x_1}, \overline{x_2}, a, \overline{b}, c\}, \{\overline{x_1}, \overline{x_2}, \overline{a}, b, \overline{c}\}$$

Observe that, for each assignment to x_1, x_2, we are either left with a Q-Horn formula (in case of all assignments except for $x_1, x_2 \mapsto 1$) or a hitting CNF formula (if $x_1, x_2 \mapsto 1$). Now we can use F' as well as any other building blocks with this general property (including blocks of arbitrary size) to construct a CNF formula F of arbitrary size by identifying individual backdoor variables inside the blocks in a tree-like pattern. For instance, consider the CNF formula F defined as follows. Let n be an arbitrary positive integer and define $var(F) = \{y_1, \ldots, y_n\} \bigcup \{a_i, b_i, c_i | 1 \leq i \leq n-1\}$. We define

$$F = \bigcup_{i=1}^{n-1} \big\{\{y_i, y_{i+1}, a_i, b_i, c_i\}, \{\overline{y_i}, y_{i+1}, \overline{a_i}, \overline{c_i}\}, \{\overline{y_i}, y_{i+1}, a_i, c_i\}, \\ \{y_i, \overline{y_{i+1}}, \overline{a_i}, \overline{b_i}, c_i\}, \{\overline{y_i}, \overline{y_{i+1}}, a_i, b_i, c_i\}, \\ \{\overline{y_i}, \overline{y_{i+1}}, a_i, \overline{b_i}, c_i\}, \{\overline{y_i}, \overline{y_{i+1}}, \overline{a_i}, b_i, \overline{c_i}\}\big\}.$$

Observe that $Y = \{y_1, \ldots, y_n\}$ is a strong backdoor of F into $\{$*Q-Horn*, *Hitting CNF formulas*$\}$ and the Y-torso graph is a path.

We now proceed to proving Theorems 2 and 3. For technical reasons, we will assume that every clause in the given CNF formula occurs exactly twice. Observe that this does not affect the satisfiability of the CNF formula or the fact that a set of variables is an acyclic strong backdoor into $\mathcal{U}$. However, it does impose certain useful structure on the incidence graph of F, as is formalized in the following observation (recall that a *cut-vertex* is a vertex whose deletion disconnects at least 2 of its neighbors). For the following, recall that $\mathsf{Inc}(F)$ denotes the incidence graph of F.

Observation 1. *Let F be a CNF formula where every clause has a duplicate. Then, every cut-vertex of the graph $\mathsf{Inc}(F)$ corresponds to a variable-vertex.*

Proof. Let c and c' be the clause-vertices in $\mathsf{Inc}(F)$ corresponding to a clause and its duplicate. Observe that if c were a cut-vertex, then deleting it must disconnect at least two variables contained in c. But this leads to a contradiction due to the presence of the clause-vertex c'. □

The following lemma is a consequence of Proposition 3 and the fact that the size of the incidence graph of F is linear in the size of F.

Lemma 4. *There is an algorithm that takes as input a CNF formula F of size n and outputs the block decomposition of $\mathsf{Inc}(F)$ in time $\mathcal{O}(n)$.*

Henceforth, we will drop the explicit mention of F having duplicated clauses. We will also assume without loss of generality that $\mathsf{Inc}(F)$ is connected. Let (T, η)

be the block decomposition of $\mathsf{Inc}(F)$. For every $t \in T$, we define the tree β_t as the subtree of T rooted at t and we denote by γ_t the subformula of F induced by the clauses whose corresponding vertices are contained in the set $\bigcup_{t' \in V(\beta_t)} \eta(t')$.

We provide a useful observation which allows us to move freely between speaking about the CNF formula and its incidence graph. For a graph G and $X \subseteq V(G)$, we denote by $\mathsf{Torso}_G(X)$ the graph defined over the vertex set X as follows: for every pair of vertices $x_1, x_2 \in X$, we add the edge (x_1, x_2) if (a) $(x_1, x_2) \in E(G)$ or (b) x_1 and x_2 both have a neighbor in the same connected component of $G - X$.

Observation 2. *Let F be a CNF formula, $G = \mathsf{Inc}(F)$ and $X \subseteq var(X)$. Then, there is an edge between x_1 and x_2 in the X-torso graph G_F^X if and only if there is an edge between x_1 and x_2 in the graph $\mathsf{Torso}_G(X)$.*

The following lemma formalizes the crucial structural observation on which our algorithm is based.

Lemma 5. *Let F be a CNF formula and let $S \subseteq var(F)$ be such that the S-torso graph is a forest. Let $G = \mathsf{Inc}(F)$. Then, the following statements hold.*

1. *No block of G contains more than two variables of S.*
2. *In the rooted block decomposition $(T, \eta(T))$ of G, no block vertex has three distinct children x_0, x_1, x_2 such that $var(\gamma_{x_i})$ intersects S for every $i \in \{0, 1, 2\}$.*

Proof. Due to Observation 2, it follows that the graph $\mathsf{Torso}_G(S)$ is acyclic. Now, suppose that the first statement does not hold and let v_0, v_1, v_2 be variables of S contained in the same block of G. We prove that for every $i \in \{0, 1, 2\}$, there is a pair of internally vertex-disjoint v_i-$v_{i+1\ (mod 3)}$ and v_i-$v_{i+2\ (mod 3)}$ paths in $\mathsf{Torso}_G(S)$. This immediately implies that the graph $\mathsf{Torso}_G(S)$ contains a cycle; indeed, the existence of vertex-disjoint v_0-v_1 and v_0-v_2 paths would mean that any v_1-v_2 path disjoint from v_0 guarantees a cycle, and the existence of vertex-disjoint v_1-v_2 and v_1-v_0 paths means that such a v_1-v_2 path must exist. We only present the argument for $i = 0$ because the other cases are analogous.

Observe that since v_0, v_1, v_2 are in the same block of G, there are paths P and Q in G where P is a v_0-v_1 path, Q is a v_0-v_2 path and P and Q are internally vertex-disjoint. Let $p_1, \ldots, p_s$ be the vertices of S which appear (in this order) when traversing P from v_0. If no such vertex appears as an internal vertex of P then we set $p_1 = p_s = v_1$. Similarly, let $q_1, \ldots, q_r$ be the vertices of S which appear when traversing Q from v_0 and if no such vertex appears as an internal vertex of Q, then we set $q_1 = q_r = v_2$.

It follows from the definition of $\mathsf{Torso}_G(S)$ that if $s = 1$ ($r = 1$), then there are edges (v_0, v_1) and (v_0, v_2) in this graph. Otherwise, there are edges $(v_0, p_1), (p_1, p_2), \ldots, (p_s, v_1)$ and edges $(v_0, q_1), (q_1, q_2), \ldots, (q_r, v_2)$. In either case, we have obtained a pair of internally vertex-disjoint paths in $\mathsf{Torso}_G(S)$; one from v_0 to v_1 and the other from v_0 to v_2. This completes the argument for the first statement.

The proof for the second statement proceeds along similar lines. Suppose to the contrary that there are three cut-vertices x_0, x_1, x_2 which are children of a

block-vertex b such that $var(\gamma_{x_i})$ intersects S for every $i \in \{0,1,2\}$ and let v_i be a variable of S chosen arbitrarily from γ_{x_i} (see Fig. 1). Observe that there are paths P_0, P_1, P_2 in G such that for every $i \in \{0,1,2\}$, the path P_i is a v_i-x_i path which is vertex-disjoint from any vertex (variable or clause) of G in $\gamma_{x_{i+1\ (mod3)}}$ or $\gamma_{x_{i+2\ (mod3)}}$. Without loss of generality, we assume that v_i is the *only* vertex of S on the path P_i.

Now, following the same argument as that for the first statement, the paths P_0, P_1, P_2 and the fact that x_0, x_1, x_2 are contained in the same block of G together imply that $\mathsf{Torso}_G(S)$ has a pair of internally vertex-disjoint v_i-$v_{i+1\ (mod3)}$ and v_i-$v_{i+2\ (mod3)}$ paths for every $i \in \{0,1,2\}$. This in turn implies that v_0, v_1, v_2 are in the same block of $\mathsf{Torso}_G(S)$. Hence, we conclude that $\mathsf{Torso}_G(S)$ is not acyclic, a contradiction. □

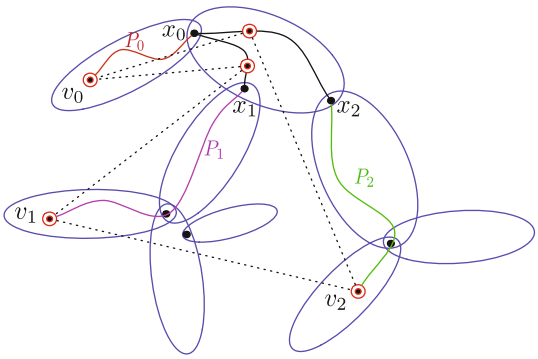

Fig. 1. An illustration of the configuration in the second statement of Lemma 5. The vertices denoted as concentric circles correspond to S. The dotted edges are edges of $\mathsf{Torso}_G(S)$.

We are now ready to present the proofs of Theorems 2 and 3. Since the proofs of both theorems are similar, we present them together. In what follows, we let $\mathcal{U} = \{\mathcal{F}_1, \ldots, \mathcal{F}_r\}$ be a set of permissively tractable classes and let $\mathcal{U}^\star = \{\mathcal{F}_1^\star, \ldots, \mathcal{F}_q^\star\}$ be a set of #-permissively tractable classes. For each $i \in \{1, \ldots, r\}$, we let $\mathcal{A}_i$ denote a polynomial time algorithm that certifies that $\mathcal{F}_i$ is permissively tractable and for each $i \in \{1, \ldots, q\}$, we let $\mathcal{A}_i^\star$ denote a polynomial time algorithm that certifies that $\mathcal{F}_i^\star$ is permissively tractable. Finally, let F be a CNF formula such that $\mathsf{btw}_{\mathcal{U}}(F) = \mathsf{btw}_{\mathcal{U}^\star}(F) = 1$.

Proof (of Theorems 2 and 3). Let S be a hypothetical strong backdoor of F into $\mathcal{U}$ ($\mathcal{U}^\star$) such that the S-torso graph is acyclic. We first handle the case when $|S| \leq 2$. In this case, we simply go over all possible pairs of variables and by assuming that they form a strong backdoor of F into $\mathcal{U}$ (respectively $\mathcal{U}^\star$), go over all instantiations of this pair of variables and independently solve (count the satisfying assignments of) each distinct connected component of the

resulting CNF formula. For this, we make use of the polynomial time algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_r$ and $\mathcal{A}_1^\star, \ldots, \mathcal{A}_q^\star$ respectively.

This step takes polynomial time and if $|S| \leq 2$, then for *some* guess of a pair of variables, we will be able to correctly determine whether F is satisfiable (correctly compute the number of satisfying assignments of F). In the case when for every pair $x, y \in var(F)$ there is an assignment $\tau : \{x, y\} \to \{0, 1\}$ and a connected component of $F[\tau]$ which is found to be not in any class of $\mathcal{U}$ (respectively $\mathcal{U}^\star$), it must be the case that $|S| > 2$.

Since S must have at least 3 variables, it follows from Lemma 5 (1) that S cannot be contained in a single block of $\mathsf{Inc}(F)$, which implies that $\mathsf{Inc}(F)$ has at least 2 distinct blocks. We now invoke Lemma 4 to compute (T, η), the block decomposition of $\mathsf{Inc}(F)$ (recall that by our assumption, $\mathsf{Inc}(F)$ is connected) and pick an arbitrary cut-vertex as the root for T. We will now execute a bottom up parse of T and use dynamic programming to test whether F is satisfiable and count the number of satisfying assignments of F. The high-level idea at the core of this dynamic programming procedure is that, by Lemma 5, at each block and cut-vertex we only need to consider constantly many vertices from the backdoor; by "guessing" these (i.e., brute-forcing over all possible choices of these), we can dynamically either solve or compute the number of satisfying assignments for the subformula "below them" in the rooted block decomposition.

For every $t, b \in V(T)$ such that t is a cut-vertex, b is a child of t and $i \in \{0, 1\}$, we define the function $\delta_i(t, b) \to \{0, 1\}$ as follows: $\delta_0(t, b) = 1$ if $\gamma_b[t \mapsto 0]$ is satisfiable and $\delta_0(t, b) = 0$ otherwise. Similarly, $\delta_1(t, b) = 1$ if $\gamma_b[t \mapsto 1]$ is satisfiable and $\delta_1(t, b) = 0$ otherwise. Finally, for every $t \in V(T)$ such that t is a cut-vertex, we define the function $\alpha_i(t) \to \{0, 1\}$ as follows. $\alpha_0(t) = 1$ if $\gamma_t[t = 0]$ is satisfiable and $\alpha_0(t) = 0$ otherwise. Similarly, $\alpha_1(t) = 1$ if $\gamma_t[t = 1]$ is satisfiable and $\alpha_1(t) = 0$ otherwise. Clearly, the formula is satisfiable if and only if $\alpha_1(\hat{t}) = 1$ or $\alpha_0(\hat{t}) = 1$, where $\hat{t}$ is the root of T.

Similarly, for every $t, b \in V(T)$ such that t is a cut-vertex, b is a child of t and $i \in \{0, 1\}$, we define $\delta_i^\star(t, b)$ to be the number of satisfying assignments of the CNF formula $\gamma_b[t = i]$. For every $t \in V(T)$ such that t is a cut-vertex and $i \in \{0, 1\}$, we define $\alpha_i^\star(t)$ to be the number of satisfying assignments of $\gamma_t[t = i]$. Clearly, the number of satisfying assignments of F is $\alpha_0^\star(\hat{t}) + \alpha_1^\star(\hat{t})$, where $\hat{t}$ denotes the root of T.

Due to Observation 1, every cut-vertex corresponds to a variable and hence the functions δ and $\delta^\star$ are well-defined. We now proceed to describe how we compute the functions δ, $\delta^\star$, α, and $\alpha^\star$ at each vertex of T assuming that the corresponding functions have been correctly computed at each child of the vertex.

We begin with the leaf vertices of T. Let b be a leaf in T. We know that b corresponds to a block of $\mathsf{Inc}(F)$ and it follows from Lemma 5 (1) that S contains at most 2 variables of γ_b. Let $Z_b = S \cap var(\eta(b))$. We guess (i.e., branch over all possible choices for) the set Z_b and for every $\tau : Z_b \cup \{t\} \to \{0, 1\}$, we run the algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_r$ (respectively $\mathcal{A}_1^\star, \ldots, \mathcal{A}_q^\star$) on the CNF formula $\gamma_b[\tau]$ to decide whether $\gamma_b[\tau]$ is satisfiable or unsatisfiable (respectively count the satisfying assignments of $\gamma_b[\tau]$) or it is not in $\mathcal{F}$ for any $\mathcal{F} \in \mathcal{U}$. By going

over all possible partial assignments $\tau : Z_b \cup \{t\} \to \{0,1\}$ in this way, we can compute $\delta_i(t,b)$ and $\delta_i^\star(t,b)$ for $i \in \{0,1\}$. Hence, we may assume that we have computed the functions $\delta_i(t,b)$ and $\delta_i^\star(t,b)$ for $i \in \{0,1\}$ for every leaf b. We now proceed to describe the computation of $\alpha, \alpha^\star$, δ and $\delta^\star$ for the internal nodes of the tree.

Let t be a cut-vertex in T such that $\delta_i(t,b)$ ($\delta_i^\star(t,b)$) has been computed for every $i \in \{0,1\}$ and $b \in \mathsf{child}(t)$. Then, $\alpha_i(t)$ is simply defined as $\bigwedge_{b \in \mathsf{child}(t)} \delta_i(t,b)$ for each $i \in \{0,1\}$. On the other hand, for each $i \in \{0,1\}$ $\alpha_i^\star(t)$ is defined as $\prod_{b \in \mathsf{child}(t)} \delta_i(t,b)$.

Finally, let b be a block-vertex in T such that for every $i \in \{0,1\}$, the value $\alpha_i(t)$ ($\alpha_i^\star(t)$) has been computed for every child t of b. Let t^* be the parent of b in T. It follows from Lemma 5 (2) that for at most 2 children $t_1, t_2 \in \mathsf{child}(b)$, the CNF formulas γ_{t_1} and γ_{t_2} contain a variable of S. Furthermore, it follows from Lemma 5 (1) that at most 2 variables of S are contained in $\eta(b)$. This implies that the CNF formula $\gamma_b \backslash (\gamma_{t_1} \cup \gamma_{t_2})$ has a strong backdoor of size at most 2 into $\mathcal{U}$ (respectively $\mathcal{U}^\star$). Hence, we can simply guess the set $Z = \{t_1, t_2\} \cup (S \cap \eta(b))$ which has size at most 4. We can then use the polynomial time algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_r$ ($\mathcal{A}_1^\star, \ldots, \mathcal{A}_q^\star$) to solve (count the satisfying assignments of) the CNF formula $(\gamma_b \backslash (\gamma_{t_1} \cup \gamma_{t_2}))[\tau]$ for every partial assignment $\tau : Z \cup \{t^*\} \to \{0,1\}$ and along with the pre-computed functions $\alpha_i(t_j)$, $\alpha_i^\star(t_j)$ for $i \in \{0,1\}, j \in \{1,2\}$, compute $\delta_p(t^*,b)$ and $\delta_p^\star(t^*,b)$ for each $p \in \{0,1\}$. While computing $\delta_p(t^*,b)$ is straightforward, note that $\delta_p^\star(t^*,b)$ is defined as $\sum_{\tau : \tau(t^*)=p} (\alpha_{\tau(t_1)}(t_1) \cdot \alpha_{\tau(t_2)}(t_2) \cdot \ell_\tau)$, where ℓ_τ is the number of satisfying assignments of $(\gamma_b \backslash (\gamma_{t_1} \cup \gamma_{t_2}))[\tau]$.

The only remaining technical subtlety in the case of counting satisfying assignments is the following. If b has exactly one child, then t_2 is left undefined and we call the unique child t_1 and work only with it in the definition of $\delta_p(b,t^*)$. In other words, we remove the term $\alpha_{\tau(t_2)}(t_2)$ from the definition of $\delta_p^\star(t^*,b)$. On the other hand, if b has at least two children but there is exactly one $t \in \mathsf{child}(b)$ such that γ_t contains a variable of S, then we set $t_1 = t$ and t_2 to be an arbitrary child of b distinct from t. Finally, if b has at least two children and there is no $t \in \mathsf{child}(b)$ such that γ_t contains a variable of S, then we define t_1 and t_2 to be an arbitrary pair of children of b. Since the set of possibilities has constant size, we can simply iterate over all of them.

Since we go over a constant number (at most 2^5) of partial assignments of $Z \cup \{t^*\}$, we will execute the algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_r$ ($\mathcal{A}_1^\star, \ldots, \mathcal{A}_q^\star$) only a constant number of times each. Therefore, this step also takes polynomial time, and the algorithm as a whole runs in polynomial time. This completes the proof of both theorems. □

5 Conclusions

We have introduced the notion of backdoor treewidth in the context of SAT and developed algorithms for deciding the satisfiability of formulas of small backdoor treewidth: (1) a fixed-parameter tractability result for backdoor treewidth with respect to Horn, Anti-Horn, and 2CNF, and (2) a polynomial-time result for

backdoor treewidth 1 with respect to a wide range or archipelagos of tractability. Both results significantly extend the borders of tractability for SAT. Our work also points to several avenues for interesting future research. In particular, our first result raises the question of whether there are further tractable classes w.r.t. which backdoor treewidth allows fixed-parameter tractability of SAT. Our second result provides a promising starting point towards the goal of obtaining a polynomial time algorithm for SAT (and #-SAT) for *every fixed* value of the backdoor treewidth with respect to a set of permissively tractable classes.

References

1. Arnborg, S., Courcelle, B., Proskurowski, A., Seese, D.: An algebraic theory of graph reduction. In: Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) Graph Grammars 1990. LNCS, vol. 532, pp. 70–83. Springer, Heidelberg (1991). doi:10.1007/BFb0017382
2. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)
3. Bodlaender, H.L., Fluiter, B.: Reduction algorithms for constructing solutions in graphs with small treewidth. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 199–208. Springer, Heidelberg (1996). doi:10.1007/3-540-61332-3_153
4. Bodlaender, H.L., van Antwerpen-de Fluiter, B.: Reduction algorithms for graphs of small treewidth. Inf. Comput. **167**(2), 86–119 (2001)
5. Bodlaender, H.L., Hagerup, T.: Parallel algorithms with optimal speedup for bounded treewidth. In: Fülöp, Z., Gécseg, F. (eds.) ICALP 1995. LNCS, vol. 944, pp. 268–279. Springer, Heidelberg (1995). doi:10.1007/3-540-60084-1_80
6. Boros, E., Hammer, P.L., Sun, X.: Recognition of q-Horn formulae in linear time. Discr. Appl. Math. **55**(1), 1–13 (1994)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009). http://mitpress.mit.edu/books/introduction-algorithms
8. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer, Cham (2015). doi:10.1007/978-3-319-21275-3
9. Diestel, R.: Graph Theory. Graduate Texts in Mathematics, vol. 173, 4th edn. Springer, Heidelberg (2012). doi:10.1007/978-3-662-53622-3
10. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer, London (2013). doi:10.1007/978-1-4471-5559-1
11. Fellows, M.R., Langston, M.A.: An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations (extended abstract). In: FOCS, pp. 520–525 (1989)
12. de Fluiter, B.: Algorithms for graphs of small treewidth. Ph.D. thesis, Utrecht University (1997)
13. Fomin, F.V., Lokshtanov, D., Misra, N., Ramanujan, M.S., Saurabh, S.: Solving d-SAT via backdoors to small treewidth. In: Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, pp. 630–641, 4–6 January 2015 (2015)

14. Ganian, R., Ramanujan, M.S., Szeider, S.: Combining treewidth and backdoors for CSP. In: 34th Symposium on Theoretical Aspects of Computer Science (STACS 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 66, pp. 36:1–36:17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
15. Ganian, R., Ramanujan, M.S., Szeider, S.: Discovering archipelagos of tractability for constraint satisfaction and counting. ACM Trans. Algorithms **13**(2), 29:1–29:32 (2017). http://doi.acm.org/10.1145/3014587
16. Gaspers, S., Szeider, S.: Backdoors to satisfaction. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond. LNCS, vol. 7370, pp. 287–317. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30891-8_15
17. Grohe, M., Kawarabayashi, K., Marx, D., Wollan, P.: Finding topological subgraphs is fixed-parameter tractable. In: Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, pp. 479–488, 6–8 June 2011
18. Hopcroft, J.E., Tarjan, R.E.: Efficient algorithms for graph manipulation [H] (algorithm 447). Commun. ACM **16**(6), 372–378 (1973)
19. Kleine Büning, H., Kullmann, O.: Minimal unsatisfiability and autarkies, Chap. 11. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 339–401. IOS Press, Amsterdam (2009)
20. Kleine Büning, H., Zhao, X.: Satisfiable formulas closed under replacement. In: Kautz, H., Selman, B. (eds.) Proceedings for the Workshop on Theory and Applications of Satisfiability. Electronic Notes in Discrete Mathematics, vol. 9. Elsevier Science Publishers, North-Holland (2001)
21. Kloks, T. (ed.): Treewidth: Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994). doi:10.1007/BFb0045375
22. Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to Horn and binary clauses. In: Proceedings of Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004), Vancouver, BC, Canada, pp. 96–103, 10–13 May 2004
23. Ordyniak, S., Paulusma, D., Szeider, S.: Satisfiability of acyclic and almost acyclic CNF formulas. Theor. Comput. Sci. **481**, 85–99 (2013)
24. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms **7**(3), 309–322 (1986)
25. Samer, M., Szeider, S.: Fixed-parameter tractability, Chap. 13. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, pp. 425–454. IOS Press, Amsterdam (2009)
26. Samer, M., Szeider, S.: Algorithms for propositional model counting. J. Discrete Algorithms **8**(1), 50–64 (2010)
27. Samer, M., Szeider, S.: Constraint satisfaction with bounded treewidth revisited. J. Comput. Syst. Sci. **76**(2), 103–114 (2010)
28. Williams, R., Gomes, C., Selman, B.: On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In: Informal Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), S. Margherita Ligure - Portofino, Italy, pp. 222–230, 5–8 May 2003

http://www.springer.com/978-3-319-66262-6

Theory and Applications of Satisfiability Testing – SAT 2017
20th International Conference, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings
Gaspers, S.; Walsh, T. (Eds.)
2017, XIII, 476 p. 68 illus., Softcover
ISBN: 978-3-319-66262-6