Recursive Backdoors for SAT

Nikolas Mählmann
University of Bremen, Germany

Sebastian Siebertz
University of Bremen, Germany

Alexandre Vigny
University of Bremen, Germany

Abstract

A strong backdoor in a formula φ of propositional logic to a tractable class $\mathscr{C}$ of formulas is a set B of variables of φ such that every assignment of the variables in B results in a formula from $\mathscr{C}$. Strong backdoors of small size or with a good structure, e.g. with small backdoor treewidth, lead to efficient solutions for the propositional satisfiability problem SAT.

In this paper we propose the new notion of *recursive backdoors*, which is inspired by the observation that in order to solve SAT we can independently recurse into the components that are created by partial assignments of variables. The quality of a recursive backdoor is measured by its *recursive backdoor depth*. Similar to the concept of backdoor treewidth, recursive backdoors of bounded depth include backdoors of unbounded size that have a certain treelike structure. However, the two concepts are incomparable and our results yield new tractability results for SAT.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Propositional satisfiability SAT, Backdoors, Parameterized Algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.73

Related Version *Previous Version*: `https://arxiv.org/abs/2102.04707`

Funding This research is funded by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) – Projektnummer 444419611.

1 Introduction

The problem of checking whether a formula of propositional logic in conjunctive normal form (CNF) is satisfiable (SAT) is one of the most central problems in computer science. The problem is often seen as the canonical NP-complete problem [1] and conjectured to be not solvable in sub-exponential time [10]. Despite this theoretical hardness result, state-of-the-art SAT solvers are able to efficiently solve multi-million variable instances arising from real-world applications. We refer to the recent survey of Ganesh and Vardi [4], who try to explain this "unreasonable effectiveness of SAT solvers". SAT is known to be solvable in polynomial time on several restricted classes of formulas, e.g. on Horn and 2CNF formulas. However, this classification falls short of explaining the practical efficiency of SAT solvers, as many efficiently solvable instances do not belong to any of these classes.

Parameterized complexity theory offers a refined view on the complexity of problems. Instead of measuring complexity only with respect to the input size n, one or more parameters are taken into account. Optimally, one can establish fixed-parameter tractability with respect to a parameter k, that is, a running time of $f(k) \cdot n^c$ for some computable function f and a constant c. In case the parameter k is small on a given class of instances, this may lead to efficient algorithms even if the inputs are large. Even though SAT solvers may not be explicitly tailored to use these parameters, it is conceivable that they implicitly exploit the structure that is imposed by them. This poses the question of parametric characterizations of real-world application instances that can be solved efficiently. One very important parameter to explain tractability is *treewidth*, which intuitively measures how tree-like an instance is, and which can be used to obtain fixed-parameter tractability for SAT [15].

A second very successful parametric approach was introduced by Williams et al. [17]. For a formula φ, a *strong backdoor* to a given class $\mathscr{C}$ of formulas is a set of variables of φ such that for every assignment of these variables one obtains a formula in $\mathscr{C}$. Similarly, a *weak backdoor* to $\mathscr{C}$ for a satisfiable formula is a set of variables of φ such that some assignment of these variables leads to a formula in $\mathscr{C}$. These notions elegantly allow to lift tractability results from classes $\mathscr{C}$ to classes that are *close* to $\mathscr{C}$. Given a formula and a strong backdoor of size k to a tractable class, one can decide satisfiability by checking 2^k tractable instances. For small k this yields efficient algorithms as noted by Nishimura et al. [11], who first studied the parameterized complexity of backdoor detection.

A lot of effort has been invested to develop fpt algorithms for backdoor detection to various tractable base classes $\mathscr{C}$, for example to classes of bounded treewidth [9] or heterogeneous classes [7]. Treewidth is a width measure for graphs that can however be applied to measure the complexity of formulas by considering the incidence graphs of formulas. The incidence graph of a formula has one vertex for each variable and one vertex for each clause. A variable vertex is connected with a clause vertex when the variable is contained positively or negatively in the clause. In the following, we will often use graph theoretic terminology for formulas, and this always refers to the incidence graph of the formula.

Apart from various base classes, alternative measures of quality of backdoors have been proposed. Backdoor trees generalize backdoor sets into decision trees, whose quality is measured by their number of leafs [14]. Recently backdoor trees have been further generalized to backdoor DNFs [12]. Ganian et al. [5] introduced the notion of backdoor treewidth, which permits fpt backdoor detection for backdoors of unbounded size. Even though backdoors of bounded treewidth can be arbitrarily large, they showed that SAT is fixed-parameter tractable when parameterized by the backdoor treewidth with respect to the classes $\mathscr{C}$ of Horn, Anti-Horn and 2CNF formulas. They also consider backdoors that split an input constraint satisfaction problem into components that may belong to different tractable classes [6]. Other recent notions of backdoors can be found in the literature such as the notions of learning-sensitive backdoors [3] and learning-sensitive backdoors with restarts [18]. For a an overview of additional works we refer to the survey by Gaspers and Szeider [8] as well as to the upcoming book chapter by Samer and Szeider [16].

In this paper we introduce the new notions of *strong* and *weak recursive backdoors* as generalizations of backdoor sets and backdoor trees. Strong recursive backdoors extend backdoor trees by not only branching on truth values but also recursively branching into the independent components of the formula that may arise after the partial assignment of variables. We measure the quality of recursive backdoors by the depth of their branching trees. The splitting into components allows recursive backdoors of bounded depth to contain an unbounded number of variables. Our definition, together with the observation that after the assignment of a variable one can independently solve the sub-instances in the arising components, reveals a new potential of backdoors for SAT.

The main power of recursive backdoors, but also the difficulty in their study, is that by assigning a variable we do not recurse into the components that are created by deleting that variable, but into the components that are created by deleting parts of the neighborhood of the variable. We show that detecting weak recursive backdoors even to the class $\mathscr{C}_0$ of edgeless incidence graphs is W[2]-hard. Our main technical contribution is an fpt algorithm that, given a formula φ and a parameter k, either decides satisfiability of φ or correctly concludes that φ has no strong recursive backdoor to $\mathscr{C}_0$ of depth at most k. Even for the class $\mathscr{C}_0$ this yields tractability results that cannot be achieved by backdoor treewidth.

We provide background in Section 2. We define recursive backdoors in Section 3 and Section 4 is devoted to a sketch of the fpt algorithm. The rest of the paper is devoted to the formal presentation and correctness proof of that algorithm. Due to space constraints we present the hardness proof only in the appended full version of the paper. Also some proofs of the main result are deferred to the appendix.

2 Preliminaries

Propositional Logic. We consider formulas of propositional logic in conjunctive normal form (CNF), represented by finite sets of clauses, and in the following when we speak of a formula we will always mean a CNF formula. We write $x, y, z\dots$ for variables and $\star, \diamond \in \{+, -\}$ for polarities. A literal is a variable with an assigned polarity. We write x_+ for the positive literal x, x_- for the negative literal $\bar{x}$, and $x_\star$ for a literal with arbitrary polarity. Every clause is a finite set of literals. We assume that no clause contains a complementary pair x_+, x_-. For a formula φ, we write $\mathrm{var}(\varphi)$ and $\mathrm{cla}(\varphi)$ to refer to the sets of variables and clauses of φ, respectively. We say that a variable x is positive (resp. negative) in a clause c if $x_+ \in c$ (resp. $x_- \in c$), and we write $\mathrm{var}(c)$ for the set of variables x with $x_\star \in c$ and $\mathrm{lit}(c)$ for the set of literals in c. For a formula φ we let $\mathrm{var}(\varphi) = \bigcup_{c\in\varphi} \mathrm{var}(c)$.

The *width* of c is $|\mathrm{var}(c)|$ and the *length* of φ is $\sum_{c\in\varphi} |\mathrm{var}(c)|$, denoted $|c|$ and $|\varphi|$, respectively. We call a clause a *d-clause* if it has width exactly d. We say that a formula has *maximal clause degree d* if each of its clauses has width at most d. We write $\mathscr{C}_d$ to refer to the class of CNF formulas with maximal clause degree d. Especially $\mathscr{C}_0$ denotes the class of empty formulas, that either contain only empty clauses or no clauses at all.

A *truth assignment* τ is a mapping from a set of variables, denoted by $\mathrm{var}(\tau)$, to $\{+, -\}$. A truth assignment τ satisfies a clause c if c contains at least one literal $x_\star$ with $\tau(x) = \star$. A truth assignment τ of $\mathrm{var}(\varphi)$ satisfies the formula φ if it satisfies all clauses of φ.

Given a formula φ and a truth assignment τ, $\varphi[\tau]$ denotes the formula obtained from φ by removing all clauses that are satisfied by τ and by removing from the remaining clauses all literals $x_\star$ with $\tau(x) \neq \star$. Note that for every formula φ and assignment τ we have $\mathrm{var}(\varphi[\tau]) \cap \mathrm{var}(\tau) = \emptyset$. If τ and τ' are assignments with $\mathrm{var}(\tau) \cap \mathrm{var}(\tau') = \emptyset$, then we write $\tau \cup \tau'$ for the unique assignment extending both τ and τ'.

Graphs. We will only consider graphs that arise as incidence graphs of formulas. The incidence graph G_φ of a formula φ is a bipartite graph with vertices $\mathrm{var}(\varphi) \cup \mathrm{cla}(\varphi)$. Slightly abusing notation we usually do not distinguish between a formula and its incidence graph. E.g. we speak of the variables and clauses of G_φ, which we denote by $\mathrm{var}(G_\varphi)$ and $\mathrm{cla}(G_\varphi)$ respectively. Vice versa, we speak e.g. of components of φ with implicit reference to the incidence graph G_φ. We drop the subscript φ if it is clear from the context. The edges of G are partitioned into two parts E_+ (positive edges) and E_- (negative edges), where a variable x is connected to a clause c by an edge $E_\star$ if $x_\star \in \mathrm{lit}(c)$. For an assignment τ we naturally define $G[\tau]$ as the incidence graph of $\varphi[\tau]$. If τ assigns only a single variable $x \mapsto \star$ we write $G[x_\star]$ for $G[\tau]$. Note that for every assignment τ, $G[\tau]$ is an induced subgraph of G. For a vertex v the closed $\star$-neighborhood of v is defined as $N_\star[v] := \{w : \{v, w\} \in E_\star\}$. For $W \subseteq V$ we write $G[W]$ for the subgraph induced by W and $G - W$ for $G[V \backslash W]$.

We refrain from formally defining treewidth and refer to the literature for background. A graph H is a minor of a graph G if H can be obtained from G by deleting edges and vertices and by contracting edges. To compare our new definition of recursive backdoors with backdoor treewidth, we mention that if a graph contains a $k \times k$ grid as a minor, then it has treewidth at least k [13].

Parameterized Complexity. A parameterized problem is called fixed-parameter tractable (fpt) if there exists an algorithm deciding the problem in time $f(k) \cdot n^c$, where n is the input size, k is the parameter, f is a computable function and c is a constant. An algorithm witnessing fixed-parameter tractability of a problem is called an fpt-algorithm for the problem.

To show that a problem is likely to not be fpt one can show that it is W[i]-hard for some $i \geqslant 1$. For this, it is sufficient to give a parameterized reduction from a known W[i]-hard problem. We refer to the book [2] for extensive background on parameterized complexity theory.

Backdoors. Let $\mathscr{C}$ be a class of formulas and let φ be a formula. A set $B \subseteq \text{var}(\varphi)$ is a *strong backdoor* of φ to $\mathscr{C}$ if for every assignment $\tau \colon B \to \{+,-\}$ the formula $\varphi[\tau]$ belongs to $\mathscr{C}$. Note that for some assignments τ the formula $\varphi[\tau]$ may not be satisfiable, even though φ is satisfiable. Hence, in the following definition of a weak backdoor we require that φ is satisfiable. If φ is satisfiable, then a set $B \subseteq \text{var}(\varphi)$ is a *weak backdoor* of φ to the class $\mathscr{C}$ if there exists an assignment $\tau \colon B \to \{+,-\}$ such that $\varphi[\tau]$ is a satisfiable formula in $\mathscr{C}$. The classical measure for the complexity or quality of a backdoor is its size.

An important recent approach to measure the complexity of a backdoor is to take its structure into account. The *treewidth of a backdoor* B is defined as the treewidth of the graph with vertex set B where two variables x and y are connected by an edge if there exists a path from a neighbor of x to a neighbor of y in $G - B$ [5]. Ganian et al. [5] also consider backdoors that split the input CNF formula into components that each may belong to a different tractable class $\mathscr{C}$.

Permissive Backdoor Detection. In their survey Gaspers and Szeider [8] differ between a strict and a permissive version of the backdoor detection problem. Given a backdoor definition $\mathcal{B}$ and a corresponding quality measure μ (e.g. strong backdoors to 2CNF measured by their size) as well as a formula φ and a parameter k. The strict backdoor detection problem, denoted as $\mathcal{B}$-Detection, asks whether or not $\mu(\varphi) \leqslant k$ holds. The permissive backdoor detection problem, denoted as SAT(μ), asks to either decide the satisfiability of φ or conclude that $\mu(\varphi) > k$ holds. The permissive version of the problem grants more freedom in algorithm design, as trivial instances can be solved without calculating the backdoor measure. However as Gaspers and Szeider point out, hardness proofs seem to be much harder for the permissive version.

3 Recursive Backdoors

Strong Recursive Backdoors. Our new concept of recursive backdoors is based on the observation that we can handle the components of $G[x_\star]$ independently whenever a variable x has been assigned. A *strong recursive backdoor* (SRB) of an incidence graph G to a class $\mathscr{C}$ is a rooted labeled tree, where every node is either labeled with a subgraph of G or with a variable in var(G). The root of the tree is labeled with G. Whenever an inner node is labeled with a connected graph H, then it has one child labeled with a variable. Whenever it is labeled with a disconnected graph, then it has one child for each of its components, labeled with the graph induced by that component. Whenever an inner node is labeled with a variable x, then its parent is labeled with a graph H, and its two children are labeled with $H[x_+]$ and $H[x_-]$, respectively. Every leaf node is labeled with a graph from $\mathscr{C}$. We call the nodes of the tree *variable nodes* or *component nodes*, according to to their labeling.

The *depth* of a strong recursive backdoor is the maximal number of variable nodes from its root to one of its leafs. The *strong recursive backdoor depth* to a class $\mathscr{C}$ ($\mathrm{srbd}_{\mathscr{C}}$) of an incidence graph G is the minimal depth of a strong recursive backdoor of G to $\mathscr{C}$. We give the following equivalent definition:

▶ **Definition 3.1** (Strong Recursive Backdoor Depth).

$$\mathrm{srbd}_{\mathscr{C}}(G) = \begin{cases} 0 & \text{if } G \in \mathscr{C} \\ 1 + \min_{x \in \mathrm{var}(G)} \max_{\star \in \{+,-\}} \mathrm{srbd}_{\mathscr{C}}(G[x_\star]) & \text{if } G \notin \mathscr{C} \text{ and } G \text{ is connected} \\ \max\{\, \mathrm{srbd}_{\mathscr{C}}(H) \;:\; H \text{ connected component of } G \,\} & \text{otherwise} \end{cases}$$

To get a better understanding of strong recursive backdoor depth we give an example of a family of incidence graphs with unbounded backdoor treewidth to 2CNF but constant strong recursive backdoor depth to $\mathscr{C}_0$, the class of edgeless graphs. For any $k \geqslant 0$, define the graph G_k as follows. We start with a $k \times k$ grid of clause vertices $\{c_{1,1}, ..., c_{k,k}\}$, depicted in yellow in Figure 1. We connect a private variable vertex to each of the corners $c_{1,1}, c_{1,k}, c_{k,1}, c_{k,k}$ of the grid. We now replace each edge of the grid by a path of length 6 (containing 5 vertices). Every second vertex on a new path is a clause vertex, connected to the two adjacent variable vertices. Furthermore, we add a special variable vertex x that is connected alternatingly with positive and negative polarity (depicted in green and blue in the figure) to the clause vertices on the new paths. Variable vertices are depicted as white vertices in the figure.

Since every clause $c_{i,j}$ is connected to at least 3 variables, every backdoor set B to 2CNF will have to contain at least one variable of every $c_{i,j}$. This implies that the B-torso of G will always contain a $k \times k$ grid as a minor, hence, will have treewidth at least k. We refer to [5, Definition 1] for the precise notions of B*-torso* and *backdoor treewidth.*

A strong recursive backdoor to $\mathscr{C}_0$ with x as its root splits every grid clause into a separate component of constant size and and therefore has constant depth.

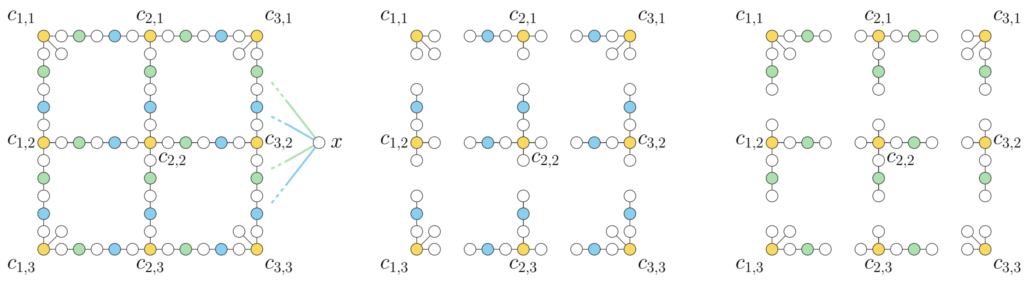

Figure 1 From left to right: G_3, $G_3[x_+]$, and $G_3[x_-]$.

Conversely, for the base class $\mathscr{C}_0$, formulas whose incidence graph is a long path, e.g. $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee \dots \vee (x_n \wedge x_{n+1})$, have constant backdoor treewidth, but unbounded strong recursive backdoor depth, as we will see in Lemma 5.2. We conclude that recursive backdoors and backdoor treewidth are incomparable.

SAT Solving and SAT Counting using Strong Recursive Backdoors. Similar to regular strong backdoor sets, strong recursive backdoors allow for polynomial time SAT Solving and SAT Counting if the backdoor is given as part of the input. First, we observe that even though it may have an unbounded branching degree, the size of a recursive backdoor is still linear in the size of its formula.

▶ **Lemma 3.2.** *Let T be a strong recursive backdoor with depth k of a formula φ to a class $\mathscr{C}$. The number of leaf nodes in T, as well as the sum of number of vertices contained in leaf nodes is bound by $2^k \cdot |\varphi|$.*

Proof. Proof by induction on k and $|\varphi|$. The bound trivially holds when $k = 0$ or $|\varphi| = 1$ and the backdoor consists of a single leaf node.

In the inductive step, a variable node increases the backdoor depth and at most doubles the number of leaves and their contained vertices, as it branches on both polarities. A component node does not increase the backdoor depth, but branches over disjoint components of strictly smaller size. As the sum of the vertices contained in the components is equal to $|\varphi|$, the number of leaves and contained vertices is again bounded by $2^k \cdot |\varphi|$. ◀

We can use this observation to construct a straight-forward bottom-up algorithm:

▶ **Proposition 3.3.** *For every class $\mathscr{C}$ where satisfiability checking (resp. counting the number of satisfying assignments) can be done in polynomial time, for every formula φ and integer k, given a strong recursive backdoor with depth k of φ to $\mathscr{C}$, we can test the satisfiability (resp. count the number of satisfying assignments) of φ in time $2^k \cdot \mathrm{poly}(|\varphi|)$.*

Proof. By Lemma 3.2, we know that we have at most $2^k \cdot |\varphi|$ instances labeling leaves, which can be solved in polynomial time. For variable nodes, the instance labeling the node is satisfiable if and only if at least one of its two children is labeled with a satisfiable instance. The number of satisfiable assignments is the sum of the satisfiable assignments for its children. For component nodes, the instance labeling the node is satisfiable if and only if all of its children are labeled with satisfiable instances. The number of satisfiable assignments is the product of the satisfiable assignments for its children. ◀

Weak Recursive Backdoors. Recall that in the definition of weak backdoors we consider only satisfiable formulas and aim to find an assignment τ that leads to a satisfiable formula $\varphi[\tau] \in \mathscr{C}$. This is also the case in the following definition of weak recursive backdoor depth:

▶ **Definition 3.4** (Weak Recursive Backdoor Depth).

$$\mathrm{wrbd}_{\mathscr{C}}(G) = \begin{cases} 0 & \textit{if } G \in \mathscr{C} \textit{ and } G \textit{ is satisfiable} \\ \infty & \textit{if } G \in \mathscr{C} \textit{ and } G \textit{ is unsatisfiable} \\ 1 + \min_{x \in \mathrm{var}(G)} \min_{\star \in \{+,-\}} \mathrm{wrbd}_{\mathscr{C}}(G[x_\star]) & \textit{if } G \notin \mathscr{C} \textit{ and } G \textit{ is connected} \\ \max \{ \mathrm{wrbd}_{\mathscr{C}}(H) \ : \ H \textit{ connected component of } G \} & \textit{otherwise} \end{cases}$$

SAT Solving using Weak Recursive Backdoors. We can use the notion of weak recursive backdoors for SAT as follows. As the existence of a weak recursive backdoor for some formula φ implies that φ is satisfiable, we do not assume that the backdoor is given with the input this time.

▶ **Proposition 3.5.** *For every class $\mathscr{C}$ for which we can test membership and satisfiability in polynomial time, for every formula φ and integer k, we can test whether φ has weak recursive backdoor depth at most k in time $(2 \cdot |\varphi|)^k \cdot poly(|\varphi|)$.*

Proof. Branch over all $2 \cdot |\varphi|$ possible truth assignments of a single variable in φ. Recurse into the components arising through the assignment, until branching depth k or a formula from $\mathscr{C}$ is reached. The leaves of the branching tree are labeled as satisfiable, if they are satisfiable members of $\mathscr{C}$, which can be tested in time $poly(|\varphi|)$. Component (resp. variable) branching nodes are labeled as satisfiable, if all (resp. any) of their children are labeled as satisfiable. It is now easy to see that $\text{wrbd}_{\mathscr{C}}(\varphi) \leqslant k$ if and only if the root node of the branching tree is labeled as satisfiable. By the same argument as in Lemma 3.2, the branching tree has at most $(2 \cdot |\varphi|)^k \cdot |\varphi|$ leaves. Therefore the presented algorithm runs in time $(2 \cdot |\varphi|)^k \cdot poly(|\varphi|)$. ◀

Note that when k is small, even this running time is a major improvement over the worst case running time of 2^{cn} implied by the exponential time hypothesis (ETH).
We will now continue with a sketch of the fpt algorithm for strong recursive backdoor detection to the class of empty formulas.

4 Proof Sketch

Our goal is to show that the permissive backdoor detection problem $\text{SAT}(\text{srbd}_{\mathscr{C}_0})$ is fixed-parameter tractable. That is, we aim to decide for a given formula φ and parameter k whether φ is satisfiable or does not have a strong recursive backdoor of depth k to the class $\mathscr{C}_0$ of empty formulas i.e. $\text{srbd}_{\mathscr{C}_0}(\varphi) > k$. Our approach is based on two main observations: Formulas with strong recursive backdoor depth k to $\mathscr{C}_0$ have both a maximal clause degree k (see Lemma 5.1) and a diameter bounded by $\lambda_k := 4 \cdot 2^k$ in each connected component (see Lemma 5.2).

We are going to design a recursive algorithm that in every step finds a bounded depth SRB that reduces the maximal clause degree of φ by one, or proves that the strong recursive backdoor depth of φ is larger than k. We extend the SRB by recursively branching on its leaves until we reach $\mathscr{C}_0$. Since φ has maximal clause degree k this yields a bounded depth SRB to $\mathscr{C}_0$ in fpt running time.

First, take a look at the special case where $G := G_\varphi$ contains a clause c of width k, i.e. a k-clause. By our first observation, the existence of c implies that $\text{srbd}_{\mathscr{C}_0}(G) \geqslant k$. Note that the degree of c can only be reduced by assigning a variable in its neighborhood.

Next, consider the case where G contains two disjoint $(k-1)$-clauses c_1, c_2 in the same connected component. Since G has limited diameter, there exists a path P of length $\leqslant \lambda_k$ between them. Since c_1 and c_2 are part of the same component we are only allowed to assign one variable to reduce the backdoor depth of that component to $k-1$. No matter which variable we choose, since c_1 and c_2 are disjoint, one of them will continue to exist in the reduced graph, which will then have backdoor depth at least $k-1$ and again, the existence of c_1, c_2, and P implies that $\text{srbd}_{\mathscr{C}_0}(G) \geqslant k$.

Given a maximal clause degree d, we generalize this strategy for arbitrary depths $k = d+j$ by searching for so called *obstruction-trees.* An obstruction-tree for depth k is a structured set of vertices, whose existence in G will guarantee that G has a strong recursive backdoor depth of at least k. We start by searching for d-clauses as obstruction-trees for depth d. We search for obstruction-trees for depth $d + j + 1$ by searching for two obstruction-trees of depth $d + j$ that have disjoint neighborhoods and are connected by a path of bounded length. A schematic depiction of obstruction-trees for maximal clause degree d and backdoor depths d, $d + 1$, and $d + 2$ is shown in Figure 2. Since the obstruction-trees are based on d-clauses we can construct an fpt algorithm that either finds an obstruction-tree or a small backdoor which reduces the maximal clause degree of G to $d - 1$.

The algorithm to find obstruction-trees, described in Proposition 5.9, is at the heart of our proof. We use this algorithm to solve G by recursively searching for obstruction-trees for depth $k + 1$. In each round we either abort and conclude that G has strong recursive backdoor depth at least $k + 1$ or reduce d and recurse until we arrive at $\mathscr{C}_0$, where we can trivially check satisfiability. If the graph splits into multiple components we can handle the components separately and aggregate the results.

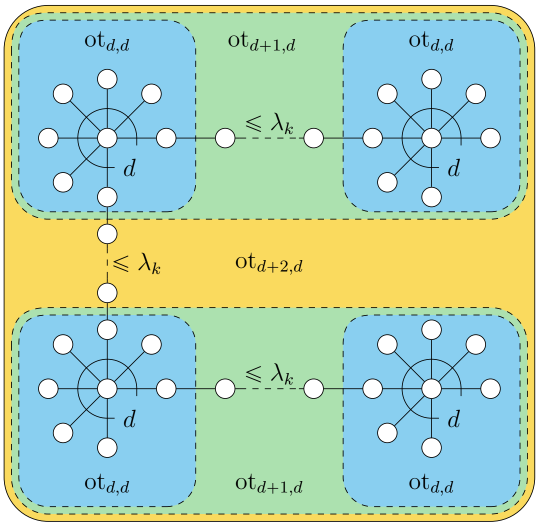

Figure 2 Schematic depiction of an obstruction-tree for maximal clause degree d and strong recursive backdoor depth $d+2$. Here, the notation "$\mathrm{ot}_{i,d}$" stands for the notion of (i, d, k)-obstruction-tree, as formally defined in Definition 5.3.

5 Permissive Strong Recursive Backdoor Detection to $\mathscr{C}_0$ Is FPT

We start by formalizing and proving the observations made in Section 4.

▶ **Lemma 5.1** (Limited Clause Degree). *For every incidence graph G and integer d, if* $\mathrm{srbd}_{\mathscr{C}_0}(G) \leqslant d$*, then G has maximal clause degree at most d, i.e. $G \in \mathscr{C}_d$.*

Proof. Proof by induction on d.
Base Case: $d = 0$. If $\mathrm{srbd}_{\mathscr{C}_0}(G) \leqslant 0$, then $G \in \mathscr{C}_0$.

Induction Step: Let d be an integer and G an incidence graph with $\mathrm{srbd}_{\mathscr{C}_0}(G) \leqslant d + 1$. Let c be any clause in G. Let H be the connected component of c in G. By assumption, $\mathrm{srbd}_{\mathscr{C}_0}(H) \leqslant d + 1$, and there must be a variable vertex x such that for every literal $x_\star$ we have $\mathrm{srbd}_{\mathscr{C}_0}(H[x_\star]) \leqslant d$. By induction, we also have that $H[x_\star] \in \mathscr{C}_d$.

Case 1: x is not connected to c. Then c is still intact in $H[x_\star]$ and by induction contains at most d variables.

Case 2: x is connected to c by an edge with polarity $+$. Then c still exists in $H[x_-]$ and by induction has degree at most d in $H[x_-]$. Therefore c contains at most $d+1$ variables in G.

Case 3: The case that x is connected to c by an edge with polarity $-$ is analogous to *Case 2.* In all cases, c contains at most $d+1$ variables in G, and therefore $G \in \mathscr{C}_{d+1}$. ◀

▶ **Lemma 5.2** (Low Diameter). *Let G be a incidence graph. If either* $\operatorname{srbd}_{\mathscr{C}_0}(G) \leqslant k$ *or* $\operatorname{wrbd}_{\mathscr{C}_0}(G) \leqslant k$, *then every connected component of G has a diameter of at most* $4 \cdot 2^k - 4$.

Proof. Proof by induction on k. For brevity we write $\operatorname{bd}(G) \leqslant k$ for the fact that either $\operatorname{wrbd}_{\mathscr{C}_0}(G) \leqslant k$ or $\operatorname{srbd}_{\mathscr{C}_0}(G) \leqslant k$.

Base Case: $k = 0$. In this case, G is edgeless, and the statement holds.

Induction Step: Assume towards a contradiction that $\operatorname{bd}(G) \leqslant k+1$ and that G has a connected component H of diameter at least $4 \cdot 2^{k+1} - 3$. Hence H contains two vertices connected by a shortest path $P = (v_1, \dots, v_m)$ with $m = 4 \cdot 2^{k+1} - 2$ (such that if v_i is a clause vertex, then v_{i+1} is a variable vertex, and if v_i is a variable vertex, then v_{i+1} is a clause vertex). Also there exists a literal $y_\star$ such that y is a variable vertex in H witnessing that $\operatorname{bd}(G) \leqslant k+1$. Since P is a shortest path from v_1 to v_m, y can only be connected to at most 2 clauses in P at distance 2 from each other. Let v_{i-1} and v_{i+1} be the two clauses connected to y (the reasoning also works with only one or zero such v_j). Now assigning $y_\star$ can split P by deleting v_{i-1} and v_{i+1}. We then have that $G[y_\star]$ still contains $(v_1, ..., v_{i-2})$ and $(v_{i+2}, ..., v_m)$ as shortest paths. One of those two paths will include at least

$$\left\lceil \frac{m-3}{2} \right\rceil = \left\lceil \frac{4 \cdot 2^{k+1} - 2 - 3}{2} \right\rceil = \left\lceil 4 \cdot 2^k - \frac{5}{2} \right\rceil = 4 \cdot 2^k - 2$$

vertices. One component of $H[y_\star]$ therefore has a diameter of at least $4 \cdot 2^k - 3$. This contradicts the fact that, by induction, $\operatorname{bd}(H[y_\star]) \leqslant k$. Therefore we have $\operatorname{bd}(G) \leqslant k+1$. ◀

In the rest of the paper, we use $\lambda_k = 4 \cdot 2^k$ for brevity.

5.1 Obstruction-Trees

We now turn to the concept of obstruction-trees. We first define them and then prove some of their properties. The main property, proved in Proposition 5.6, is that the existence of such trees witnesses a lower bound for the depth of a strong recursive backdoor to the class $\mathscr{C}_0$.

▶ **Definition 5.3** (Obstruction-Trees and Destroy Neighborhoods). *For all integers k, d and incidence graphs G in $\mathscr{C}_d$, we inductively for $i \geqslant d$ define the notion of an (i, d, k)-*obstruction-tree *T of G with* elements $V(T)$ *and* destroy-neighborhood $N^\dagger_G[T]$. *We use* $\operatorname{cla}(T)$ *and* $\operatorname{var}(T)$ *to denote the clauses and variables of $V(T)$.*
For a set $T = \{c, x_1, ..., x_d\}$, where c is a d-clause of G with $\operatorname{var}(c) = \{x_1, \dots, x_d\}$, *we have:*

1. *T is a (d, d, k)-obstruction-tree.*
2. *$V(T)$ are the elements of T.*
3. $N^\dagger_G[T] := \operatorname{var}(T) = \{x_1, \dots, x_d\}$.

Inductively, for a triple $T = (T_1, P, T_2)$ where T_1 and T_2 are (i, d, k)-obstruction-trees of G such that $N^\dagger_G[T_1] \cap N^\dagger_G[T_2] = \varnothing$, and P is a path of length at most λ_k connecting T_1 and T_2, we have:

1. *T is an $(i+1, d, k)$-obstruction-tree.*
2. *$V(T) := V(T_1) \cup V(P) \cup V(T_2)$.*
3. *$N^{\dagger}_{G}[T] := \mathrm{var}(T) \cup \{x :$ there exist $c_1, c_2 \in \mathrm{cla}(T)$with $\{x, c_1\} \in E_+$ and $\{x, c_2\} \in E_-\}$.*

Observe that $V(T)$ is a connected subset of G. We now show, that our definition of a destroy neighborhood is a small set of variables, that shields the obstruction-tree from the rest of the graph. Remember that $\lambda_k = 4 \cdot 2^k$.

▶ **Proposition 5.4** ($N^{\dagger}$ Is Small). *For all integers i, d, k with $d \leqslant k$, for every incidence graph G in $\mathscr{C}_d$, and every (i, d, k)-obstruction-tree T of G, we have $|V(T)| \leqslant 3^{i-d} \cdot \lambda_k$ and $|N^{\dagger}_{G}[T]| \leqslant 3^{i-d} \cdot \lambda_k \cdot d$.*

▶ **Proposition 5.5** ($N^{\dagger}$ Is a Destroy Neighborhood). *For all integers i, d, k, every incidence graph G in $\mathscr{C}_d$, every (i, d, k)-obstruction-tree T of G, and every variable x of G, if $x \notin N^{\dagger}_{G}[T]$, then T is also an (i, d, k)-obstruction-tree in at least one of $G[x_+]$ and $G[x_-]$.*

Due to space constraints, both proofs were moved to Appendix A.1 and Appendix A.2. We now turn to the main property of obstruction-trees, which explains why this notion is relevant in this context.

▶ **Proposition 5.6** (Obstruction-Trees Obstruct). *For all integers i, d, k and every incidence graph G in $\mathscr{C}_d$, if there is an (i, d, k)-obstruction-tree T of G, then $\mathrm{srbd}_{\mathscr{C}_0}(G) \geqslant i$.*

Proof. Proof by induction on i.
Base Case: $i = d$. Follows immediately from Lemma 5.1.

Induction Step: Let T be an $(i+1, d, k)$-obstruction-tree of G, and assume towards a contradiction that $\mathrm{srbd}_{\mathscr{C}_0}(G) < i + 1$. By Definition 5.3 we get T_1 and T_2, two (i, d, k)-obstruction-trees connected by a path P. Additionally in every connected component H of G, $\mathrm{srbd}_{\mathscr{C}_0}(H) < i + 1$ holds as well. Since $V(T)$ is connected, there exists one component H containing T. Then, there must exists a variable x in H such that $\mathrm{srbd}_{\mathscr{C}_0}(H[x_+]) < i$ and $\mathrm{srbd}_{\mathscr{C}_0}(H[x_-]) < i$. Assume $x \notin N^{\dagger}_{H}[T_1]$. Then by Proposition 5.5 we get that T_1 remains an (i, d, k)-obstruction-tree in either $H[x_+]$ or $H[x_-]$. We use the induction hypothesis to conclude that one of the graphs has strong recursive backdoor depth at least i and we get a contradiction. Assume $x \in N^{\dagger}_{H}[T_1]$. Then $x \notin N^{\dagger}_{H}[T_2]$ by Definition 5.3 and we can make the same argument. ◀

Finally, and for technical reasons, we need to show that if we assign a variable, we do not increase the strong recursive backdoor depth. Also if we find an obstruction-tree after assigning some variables, then it is also an obstruction-tree in the original graph with the same destroy neighborhood.

▶ **Lemma 5.7** ($\mathrm{srbd}_{\mathscr{C}_0}$ Is Closed Under Assignments). *For every integer k, every incidence graph G, and every literal $x_\star$ in G, if $\mathrm{srbd}_{\mathscr{C}_0}(G) \leqslant k$, then $\mathrm{srbd}_{\mathscr{C}_0}(G[x_\star]) \leqslant k$.*

▶ **Proposition 5.8** (Obstruction-Trees Can Be Lifted). *For all integers i, d, k with $d \leqslant i$ and $d \leqslant k$, every incidence graph G in $\mathscr{C}_d$, every obstruction-tree T and every literal $x_\star$ of G, if T is an (i, d, k)-obstruction-tree of $H := G[x_\star]$, then it is also an (i, d, k)-obstruction-tree of G and we have $N^{\dagger}_{G}[T] = N^{\dagger}_{H}[T]$.*

The proofs of these statements can be found in Appendix A.3 and Appendix A.4.

5.2 Algorithms

We finally turn to the algorithm. We first show that we can efficiently compute an obstruction-tree, or make progress towards computing a strong recursive backdoor to $\mathscr{C}_0$. Making progress here means decreasing the clause degree of the graph. At every step, the algorithm may stop if it concludes that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$.

▶ **Proposition 5.9** (Obstruction-Trees Are Easy to Compute)**.** *There is an algorithm that, given three integers i, d, k, with $d \leqslant i \leqslant k+1$, and an incidence graph G in $\mathscr{C}_d$, in time $2^{2^{O(k)}} \cdot |G|$ either*

1. *returns an (i,d,k)-obstruction-tree T, or*
2. *returns a strong recursive backdoor B to $\mathcal{C}_{d-1}$ of depth at most $g(i,d,k) := 3^{i-d} \cdot \lambda_k \cdot d$, or*
3. *concludes that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$.*

Proof. We fix d, k and prove the claims by induction on i and $|G|$.
Base Case: When $i = d$, we search for a clause c connected to d variables. If we find c, then c is a (d,d,k)-obstruction-tree and we return it. If there is no such clause, then G is also in $\mathscr{C}_{d-1}$, and the leaf node labeled G is a strong recursive backdoor to $\mathcal{C}_{d-1}$.

Induction Step on i: We now fix i and assume that have a working algorithm with parameters (i,d,k) for any incidence graph $G \in \mathscr{C}_d$. We now explain by induction on $|G|$ how to build an algorithm with parameters $(i+1,d,k)$.
Base Case: In the base case $|G| = 1$, then $G \in \mathscr{C}_0$, and there is nothing to do.

Induction Step on $|G|$ when G is not connected: All connected components of G have size strictly smaller than G, and we can run the algorithm with parameters $(i+1,d,k)$ on each. If in one connected component H, we find an $(i+1,d,k)$-obstruction-tree T, then T is also an $(i+1,d,k)$-obstruction-tree of G and we are done. If for one connected component H we have $\mathrm{srbd}_{\mathscr{C}_0}(H) > k$, then it also holds that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$. Finally, if for every connected component H we find a strong recursive backdoor B_H to $\mathcal{C}_{d-1}$ of depth at most $g(i+1,d,k)$, we can merge them to build a recursive backdoor B to $\mathcal{C}_{d-1}$ for G. In order to do this, we insert a root node labeled G, whose children are all the B_H. Since we do not insert a variable node, B has still depth at most $g(i+1,d,k)$.

Induction Step on $|G|$ when G is connected: In this case, we use the induction hypothesis on G with parameters (i,d,k). If the algorithm provides a strong recursive backdoor or concludes that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$, we are done. We focus on the case where the algorithm returns an (i,d,k)-obstruction-tree T_1 for G and $N^{\dagger}_G[T_1]$. We define $\mathcal{T}$ as the set of all possible truth assignments to the variables of $N^{\dagger}_G[T_1]$. For every τ in $\mathcal{T}$, we define $H_\tau := G[\tau]$. On every H_τ we run the algorithm given by induction with parameters (i,d,k).

Case 1: For at least one H_τ we have that $\mathrm{srbd}_{\mathscr{C}_0}(H_\tau) > k$. By Lemma 5.7, $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$ follows.

Case 2: For at least one H_τ we find an (i,d,k)-obstruction-tree T_2. By Proposition 5.8, T_2 is also an (i,d,k)-obstruction-tree in G and $N^{\dagger}_{H_\tau}[T_2] = N^{\dagger}_G[T_2]$. Since none of the variables in $N^{\dagger}_G[T_1]$ appear in H_τ, we have that $N^{\dagger}_G[T_1] \cap N^{\dagger}_G[T_2] = \varnothing$. We run a BFS algorithm to find a shortest path P between a vertex of T_1 and T_2 in G. If P has length greater than λ_k, we can conclude that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$ by Lemma 5.2. If P has length at most λ_k, we have that $T = (T_1, P, T_2)$ is an $(i+1,d,k)$-obstruction-tree in G.

The process of constructing T is depicted in Figure 3. In the first panel we see that we have found T_2 in the graph H_τ. H_τ is an induced subgraph of G in which the variables of τ have been assigned and which therefore does not contain T_1 (see hatched area). In the second panel we lift T_2 into G, which also contains T_1. In the third panel, all thats left to do is to combine T_1 and T_2 into T, which then is an obstruction-tree for G.

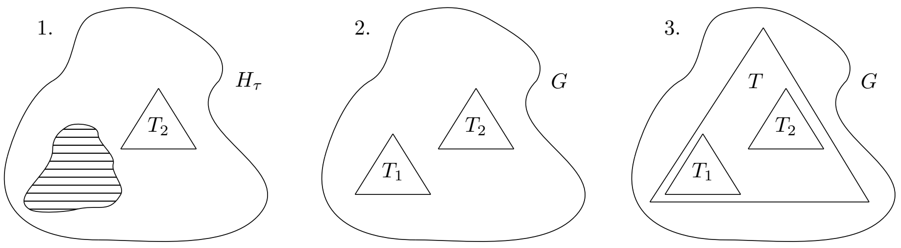

Figure 3 Searching an obstruction-tree.

Case 3: For every H_τ we find a strong recursive backdoor B_τ to $\mathscr{C}_0$ of depth at most $g(i,d,k)$. In this case, we can combine them and build a strong recursive backdoor B of G to $\mathcal{C}_{d-1}$. To do so, we start with the complete binary tree, where at each step we branch over one variable in $N^\dagger_G[T_1]$. At depth $|N^\dagger_G[T_1]|$, each node corresponds to an assignment τ of $\mathcal{T}$. We then finish the tree by plugging B_τ in the branch corresponding to τ. B is a strong recursive backdoor of G to $\mathcal{C}_{d-1}$, and its depth is bounded by: $|N^\dagger_G[T_1]| + g(i,d,k) \leqslant g(i+1,d,k)$.

The process of merging backdoors is depicted in Figure 4. The backdoors B_τ are depicted as trees, whose root nodes are merged by and form the leaves of a full binary tree over the variables of $N^\dagger_G[T_1]$.

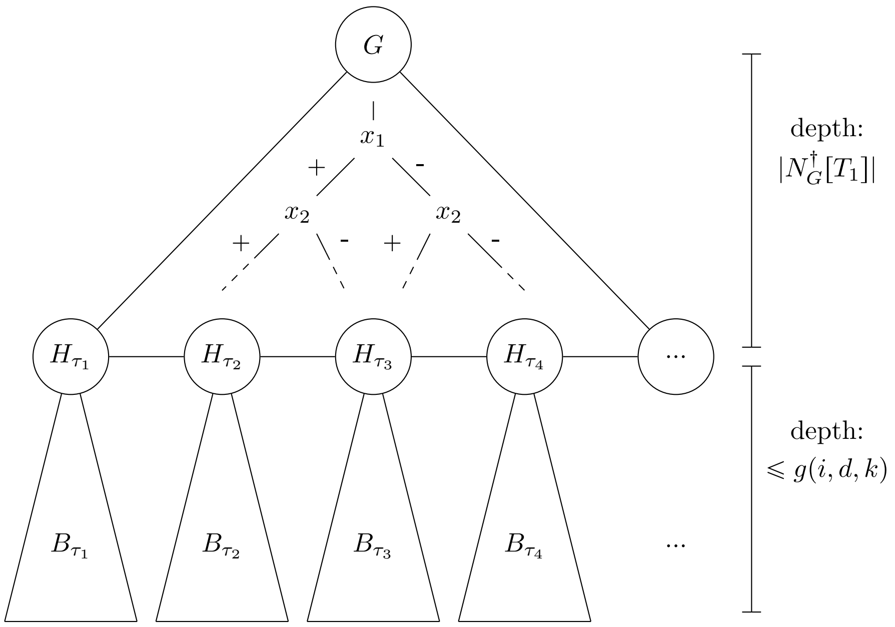

Figure 4 Merging backdoors.

Time Complexity: The proof for the time complexity can be found in Appendix A.5. ◀

We now use the result of Proposition 5.9 sufficiently many times so that the degree of the input graph reaches 0. Again, at any point, the algorithm may stop and conclude that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$. Remember that $\lambda_k = 4 \cdot 2^k$.

▶ **Theorem 5.10.** *There is an algorithm that, given as input an integer k and an incidence graph G, in time $2^{2^{\mathcal{O}(k)}} \cdot |G|$ either:*

1. *returns a strong recursive backdoor of G to $\mathscr{C}_0$ of depth at most $3^k \cdot \lambda_k \cdot k^2$, or*
2. *concludes that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$.*

Proof. Let d be the maximal degree of a clause in G. If $d > k$ conclude that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$ by Lemma 5.1. Otherwise handle G using induction on d to search for a SRB to $\mathscr{C}_0$ of depth at most $3^k \cdot \lambda_k \cdot d^2$:
Base Case: $G \in \mathscr{C}_0$ and the node labeled G is a SRB to $\mathscr{C}_0$ of depth 0.

Induction Step: $G \in \mathscr{C}_{d+1}$. Run the algorithm presented in Proposition 5.9 with parameters $(k+1, d+1, k)$ on G. If it concludes that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$, or returns a $(k+1, d+1, k)$-obstruction-tree, conclude that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$ by Proposition 5.6. If a SRB B is returned, then B will have depth at most $3^{k+1-(d+1)} \cdot \lambda_k \cdot (d+1) \leqslant 3^k \cdot \lambda_k \cdot (d+1)$ and every leaf of B will be labeled with a graph H in $\mathscr{C}_d$.

We then apply the algorithm given by the induction hypothesis to every H. If for one H we get that $\mathrm{srbd}_{\mathscr{C}_0}(H) > k$, conclude that $\mathrm{srbd}_{\mathscr{C}_0}(G) > k$ by Lemma 5.7. If for every H we get a SRB B_H to $\mathscr{C}_0$ of depth at most $3^k \cdot \lambda_k \cdot d^2$, we use the results to build a SRB to $\mathscr{C}_0$ for G. To do so, we replace the leaf labeled H in B with B_H for every H. As a result, B will be extended to be a SRB for G to $\mathscr{C}_0$ with depth at most $3^k \cdot \lambda_k \cdot (d+1)^2$.

Time Complexity: The proof for the time complexity can be found in Appendix A.6. ◀

▶ **Corollary 5.11.** *Given a formula φ and a parameter k there is an algorithm that solves $SAT(\mathrm{srbd}_{\mathscr{C}_0})$ in time $2^{2^{\mathcal{O}(k)}} \cdot |\varphi|$.*

Proof. We compute the satisfiability of φ in two steps. First run the algorithm given in Theorem 5.10 with parameters φ and k in time $2^{2^{\mathcal{O}(k)}} \cdot |\varphi|$. If the algorithm concludes that $\mathrm{srbd}_{\mathscr{C}_0}(\varphi) > k$ we are finished. Otherwise a SRB with depth at most $3^k \cdot \lambda_k \cdot k^2$ of φ to $\mathscr{C}_0$ is returned.

Second, we make use of the calculated SRB by running the algorithm described in Proposition 3.3, to determine the satisfiability of φ. The satisfiability of a formula in $\mathscr{C}_0$ can be checked in constant time: If it contains no clause, then all clauses are trivially satisfied. If it contains at least one clause, then that clause is empty and unsatisfiable. Therefore the second step runs in time $\mathcal{O}(2^{3^k \cdot \lambda_k \cdot k^2} \cdot |\varphi|)$. Adding up the running times, we get a total time complexity of $2^{2^{\mathcal{O}(k)}} \cdot |\varphi|$. ◀

6 Weak Recursive Backdoor Detection to $\mathscr{C}_0$ Is $W[2]$-Hard

In this section, we show that the parametrized problem of detecting a weak recursive backdoor of depth k to the class of edgeless graphs is $W[2]$-hard when parametrized by k.

▶ **Theorem 6.1.** *WEAK-RECURSIVE-$\mathscr{C}_0$-BACKDOOR-DETECTION is W[2]-hard.*

Due to space constraints, the proof was moved to Appendix A.7.

7 Conclusion

We have proposed the new notions of *strong* and *weak recursive backdoors*, which exploit the structure of formulas that can be recursively split into independent parts by partial assignments. Recursive backdoors are measured by their depth and can contain, even at bounded depth, an unbounded number of variables. In our work we have focused on the tractable base class of empty formulas $\mathscr{C}_0$. We have shown, that detecting weak recursive backdoors to $\mathscr{C}_0$ is W[2]-hard. Our main technical contribution is an fpt algorithm that detects strong recursive backdoors of bounded depth to $\mathscr{C}_0$. Even for $\mathscr{C}_0$ this extends tractable SAT Solving to a new class of formulas.

Our result raises the question of whether the detection of strong recursive backdoors can be expanded to larger base classes such as 2CNF or Horn. Especially 2CNF seems to be in reach, as similar to $\mathscr{C}_0$, incidence graphs of formulas with bounded recursive backdoor depth to 2CNF have a bounded clause degree. This is the first ingredient for our algorithm to $\mathscr{C}_0$. However our algorithm is limited to finding backdoors to $\mathscr{C}_0$, as the second ingredient, which is bounded incidence graph diameter, is not given when searching for backdoors to 2CNF.

References

1 Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

2 Marek Cygan, Fedor Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer Publishing Company, Incorporated, January 2015.

3 Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal. Backdoors in the context of learning. In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 73–79. Springer, 2009.

4 Vijay Ganesh and Moshe Y. Vardi. On the unreasonable effectiveness of SAT solvers. In *Beyond the Worst-Case Analysis of Algorithms*, pages 547–566. Cambridge University Press, 2020.

5 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Backdoor treewidth for sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 20–37. Springer, 2017.

6 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Transactions on Algorithms*, 13(2):1–32, 2017.

7 Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Živný. Backdoors into heterogeneous classes of SAT and CSP. *Journal of Computer and System Sciences*, 85:38–56, 2017.

8 Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In *The Multivariate Algorithmic Revolution and Beyond*, pages 287–317. Springer, 2012.

9 Serge Gaspers and Stefan Szeider. Strong backdoors to bounded treewidth SAT. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 2013.

10 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

11 N. Nishimura, P. Ragde, and Stefan Szeider. Detecting backdoor sets with respect to horn and binary clauses. In *SAT*, 2004.

12 Sebastian Ordyniak, André Schidler, and Stefan Szeider. Backdoor DNFs. Technical Report AC-TR-21-001, Algorithms and Complexity Group, TU Wien, 2021.

13 Neil Robertson and Paul D Seymour. Graph minors V. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986.

14 Marko Samer and Stefan Szeider. Backdoor trees. In *Proceedings of the 23rd National Conference on Artificial Intelligence – Volume 1*, AAAI'08, page 363–368. AAAI Press, 2008.

15 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *Journal of Discrete Algorithms*, 8(1):50–64, 2010.
16 Marko Samer and Stefan Szeider. Fixed-parameter tractability. Technical Report AC-TR-21-004, Algorithms and Complexity Group, TU Wien, 2021. Chapter 17, Handbook of Satisfiability, 2nd Edition, 2021.
17 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 1173–1178. Morgan Kaufmann Publishers Inc., 2003.
18 Edward Zulkoski, Ruben Martins, Christoph M. Wintersteiger, Robert Robere, Jia Hui Liang, Krzysztof Czarnecki, and Vijay Ganesh. Learning-sensitive backdoors with restarts. In *CP*, volume 11008 of *Lecture Notes in Computer Science*, pages 453–469. Springer, 2018.

A Omitted Proofs

A.1 Proof of Proposition 5.4

Proof. The second claim easily follows the first one. The set $N^{\dagger}_{T}[G]$ contains $\mathrm{var}(T)$ and at most the variables connected to a clause from $\mathrm{cla}(T)$. Since $G \in \mathscr{C}_d$, we get that $|N^{\dagger}_{G}[T]| \leqslant |V(T)| \cdot d$. Now we prove that $|V(T)| \leqslant 3^{i-d} \cdot \lambda_k$ by induction on i.
Base Case: T is an (d, d, k)-obstruction-tree. By definition, $|V(T)| = |T| = d + 1 \leqslant 3^{d-d} \cdot \lambda_k$.

Induction Step: T is an $(i + 1, d, k)$-obstruction-tree. Then $T = (T_1, P, T_2)$ such that T_1 and T_2 are (i, d, k)-obstruction-trees of G and $|V(T)| \leqslant |V(T_1)| + |V(P)| + |V(T_2)|$. We apply our induction hypothesis to conclude that both $V(T_1)$ and $V(T_2)$ have at most $3^{i-d} \cdot \lambda_k$ elements. P has at most λ_k elements by definition. We conclude that $|V(T)| \leqslant 3 \cdot 3^{i-d} \cdot \lambda_k = 3^{i+1-d} \cdot \lambda_k$. ◀

A.2 Proof of Proposition 5.5

In order to prove Proposition 5.5, we first prove an intermediate result:

▶ **Proposition A.1** (Obstruction-Trees Are only Influenced by Adjacent Variables). *For all integers i, d, k, every incidence graph G in $\mathscr{C}_d$, every (i, d, k)-obstruction-tree T of G, every variable x in G, and every polarity $\star$, if $x \notin \mathrm{var}(T)$ and for all $c \in \mathrm{cla}(T)$ we have $\{x, c\} \notin E_\star$, then T is still an (i, d, k)-obstruction-tree in $G[x_\star]$.*

Proof. Proof by induction on i.
Base Case: $i = d$. Then T contains a d-clause c and its adjacent variables $\mathrm{var}(T)$. If $x \notin \mathrm{var}(T)$, then T remains untouched and continues to be a (d, d, k)-obstruction-tree of $G[x_\star]$.

Induction Step: $i > d$. Then $T = (T_1, P, T_2)$, where T_1 and T_2 are (i, d, k)-obstruction-trees of G. Assume $x \notin \mathrm{var}(T)$ and for all $c \in \mathrm{cla}(T)$ we have $\{x, c\} \notin E_\star$. Since $\mathrm{cla}(T_1)$ and $\mathrm{cla}(T_2)$ are both subsets of $\mathrm{cla}(T)$ we can apply our induction hypothesis and conclude that T_1 andT_2 are still (i, d, k)-obstruction-trees of $G[x_\star]$. Since $G[x_\star]$ is a subgraph of G we know that $N^{\dagger}_{G[x_\star]}[T_1] \cap N^{\dagger}_{G[x_\star]}[T_2]$ is still empty. Since x is not contained in $\mathrm{var}(P) \subseteq \mathrm{var}(T)$ and is also not connected to a clause of $\mathrm{cla}(P) \subseteq \mathrm{cla}(T)$ by polarity $\star$, we conclude that P still is a path of the same length in $G[x_\star]$. It follows that T must be $(i + 1, d, k)$-obstruction-tree in $G[x_\star]$. ◀

We now continue with the proof of Proposition 5.5:

Proof. Assume towards a contradiction that $x \notin N^{\dagger}_{G}[T]$ and T is no (i, d, k)-obstruction-tree of $G[x_+]$ and $G[x_-]$. If T is no (i, d, k)-obstruction-tree in $G[x_+]$, then by Proposition A.1, either $x \in \mathrm{var}(T)$ or there exists a clause c_1 connected to x by a positive edge. Since the former contradicts with $x \notin N^{\dagger}_{G}[T]$, we have that c_1 exists. Now assume that T is also no (i, d, k)-obstruction-tree in $G[x_-]$. By the same reasoning conclude that there exists a clause c_2 connected to x by a negative edge. From the existence of both c_1 and c_2 connected with different polarities to x we conclude that $x \in N^{\dagger}_{G}[T]$ and get a contradiction. ◀

A.3 Proof of Lemma 5.7

Proof. Proof by induction on k.
Base Case: $k = 0$. Then G is edgeless and remains edgeless when a variable is assigned.

Induction Step: Let G be an incidence graph such that $\mathrm{srbd}_{\mathscr{C}_0}(G) \leqslant k + 1$, and let $x_\star$ be any literal of G. If G is connected, then by Definition 3.1 there exists a variable y such that $\mathrm{srbd}_{\mathscr{C}_0}(G[y_\star]) \leqslant k$. If $x = y$, then $\mathrm{srbd}_{\mathscr{C}_0}(G[x_\star]) \leqslant k + 1$ holds trivially. If $x \neq y$ then we apply our induction hypothesis and because $\mathrm{srbd}_{\mathscr{C}_0}(G[y_\star]) \leqslant k$ get that $\mathrm{srbd}_{\mathscr{C}_0}(G[y_\star, x_\star]) \leqslant k$. This leads to $\mathrm{srbd}_{\mathscr{C}_0}(G[x_\star, y_\star]) \leqslant k$, which again implies that $\mathrm{srbd}_{\mathscr{C}_0}(G[x_\star]) \leqslant k + 1$ holds. If G contains multiple components, then the same argument applies for the component that contains x and the other components remain unchanged. ◀

A.4 Proof of Proposition 5.8

Proof. Proof by induction on i.
Base Case: $i = d$. Then T contains a d-clause c and its variables $x_1, \ldots, x_d$ in H and $N^{\dagger}_{H}[T]$ contains all x_i. Since G has maximal clause degree d, c must also be a d-clause in G with the same neighborhood.

Induction Step: Assume T is an $(i+1, d, k)$ obstruction-tree of H. Then $T = (T_1, P, T_2)$ such that T_1 and T_2 are (i, d, k)-obstruction-trees of H, and P is a path of length at most λ_k. By applying the induction hypothesis, we get that T_1 and T_2 are also (i, d, k)-obstruction-trees of G and that $N^{\dagger}_{G}[T_1] = N^{\dagger}_{H}[T_1]$ and $N^{\dagger}_{G}[T_2] = N^{\dagger}_{H}[T_2]$ are disjoint. P obviously still is a path of length at most λ_k in G, so T is indeed an $(i + 1, d, k)$-obstruction-tree of G.

We now show that $N^{\dagger}_{H}[T] = N^{\dagger}_{G}[T]$. Since H is an induced subgraph of G, we get that $N^{\dagger}_{H}[T] \subseteq N^{\dagger}_{G}[T]$. To show $N^{\dagger}_{H}[T] \supseteq N^{\dagger}_{G}[T]$, pick any variable y from $N^{\dagger}_{G}[T]$. If $y \in \mathrm{var}(T)$ we get that $y \in N^{\dagger}_{H}[T]$ by definition. If y is positively connected to c_1 and negatively connected to c_2 for two clauses $c_1, c_2 \in \mathrm{cla}(T)$, then $y \neq x$, since otherwise the assignment of y in H would delete a clause from T, which contradicts the fact that T is an $(i + 1, d, k)$-obstruction-tree in H. Since y is not equal to x, its edges to c_1 and c_2 are not affected by the assignment of x in H and again $y \in N^{\dagger}_{H}[T]$ holds. It follows that $N^{\dagger}_{G}[T] = N^{\dagger}_{H}[T]$. ◀

A.5 Time Complexity of Proposition 5.9

Proof. Let us prove by induction that the time complexity of the algorithm presented in Proposition 5.9 is $2^{2^{O(k)}} \cdot |G|$. This clearly holds when $|G| = 1$, or when $i = d$. We now move on to the induction and analyze the run of the algorithm with parameters $(i + 1, d, k)$ on a graph G.

First, note that when we split among several connected components these components are disjoints. The sum of the sizes of these components is the size of the G. Unifying the recursive backdoor, given by the components, by adding a common root node takes at most linear time, which is consistent with our hypothesis.

Second, when the graph is connected we first run the algorithm with parameters (i, d, k). If the run does not stop there, we have an (i, d, k)-obstruction-tree T. We then consider a number of truth assignments that is bounded by $2^{N_G^\dagger[T]}$. For each of these assignment, we run again our algorithm with parameters (i, d, k), on graphs smaller than G.

If we find a second (i, d, k)-obstruction-tree we then only need to compute shortest path, which can be performed in linear time. If every truth assignment provides a backdoor-tree, plugging them together only takes time linear in the number of possible truth assignments.

All together the procedure stays linear and the constant factor gets multiplied by a factor of the form $O\left(2^{N_G^\dagger[T]}\right)$ each time i decreases by one. By Proposition 5.4, this is bounded by $O\left(2^{3^{i-d}\cdot\lambda_k\cdot d}\right)$. Using that both $d \leqslant k$ and $i \leqslant k+1$, this is of the form $2^{2^{O(k)}}$. As i can decrease by one at most $i-d$ many times (and therefore at most i many times) until we get to a base case, we get that the final constant factor is of the form $\left(2^{2^{O(k)}}\right)^i = 2^{i2^{O(k)}} = 2^{2^{O(k)}}$.

We finally have that the overall complexity of a run with parameters (i, d, k) on a graph G is bounded by $2^{2^{O(k)}} \cdot |G|$. ◀

A.6 Time Complexity of Theorem 5.10

Proof. Let $f(k) \cdot |G|$ be the time complexity of Proposition 5.9 and $g(k, d)$ be $3^k \cdot \lambda_k \cdot d$. We prove the time complexity of $2^{g(k,d)\cdot d} \cdot f(k) \cdot |G|$ by induction on d. If $d = 0$ then we only have to construct a single node, which can be done in constant time. For graphs in $\mathscr{C}_{d+1}$, running the algorithm of Proposition 5.9 can be done in time $f(k) \cdot |G|$. If we do not find a SRB, we can abort. Otherwise we find a SRB of depth at most $g(k, d+1)$ such that all its leaves are members of $\mathscr{C}_d$. We can apply our induction hypothesis and assume that for a single leaf H, we can finish in time $2^{g(k,d)\cdot d} \cdot f(k) \cdot |H|$. Since the sum of the number of vertices in all leafs of the backdoor is at most $2^{g(k,d+1)} \cdot |G|$, we get that full algorithm has a running time in

$$f(k) \cdot |G| + 2^{g(k,d)\cdot d} \cdot f(k) \cdot 2^{g(k,d+1)} \cdot |G| \leqslant 2^{g(k,d+1)\cdot(d+1)} \cdot f(k) \cdot |G|.$$

Since both $2^{g(k,d)\cdot d}$ and $f(k)$ are in $2^{2^{\mathcal{O}(k)}}$, the overall time complexity of the algorithm is in $2^{2^{\mathcal{O}(k)}} \cdot |G|$. ◀

A.7 Proof of Theorem 6.1

Proof. We are going to show the W[2]-hardness of Weak-Recursive-$\mathscr{C}_0$-Backdoor-Detection (WR-$\mathscr{C}_0$-BD) by reduction from the W[2]-complete Set Cover problem [2, Theorem 13.28]. An instance I of the Set Cover problem is composed of a universe U, an integer k, and a set $S \subseteq P(U)$. I is a yes-instance if there exists a subset L of S with size at most k, such that the union of all sets in L is equal to U.

We reduce $I = (S, U, k)$ to the WR-$\mathscr{C}_0$-BD instance $(\varphi, k+1)$, where φ is a CNF formula over the variables $\{b_1, ..., b_{k+2}, s_1, ..., s_n\}$, where $n = |S|$. This formula is constructed in the following way:

For each element of the universe $u \in U$ a corresponding clause σ_u is created, which we call *element-clauses*. For each set S_i a corresponding variable vertex s_i is created, which we call *set-variables*. Then s_i and σ_u are connected by a positive edge when $u \in S_i$. Therefore in the incidence graph, s_i dominates all the clauses whose corresponding elements are contained in S_i.

In addition, we create $k + 2$ fresh variables b_i. Each are individually and positively connected to a fresh clause β_i. Furthermore, all b_i are negatively connected to all σ_u, creating a $K_{k+2,|U|}$ bi-clique. More formally, we have:

$$\begin{aligned}
\beta_i &:= b_i \\
\sigma_u &:= \bigvee_{i=1}^{k+2} \neg b_i \vee \bigvee_{\{i \leqslant n \,:\, u \in S_i\}} s_i \\
\varphi &:= \bigwedge_{i=1}^{k+2} \beta_i \wedge \bigwedge_{u \in U} \sigma_u
\end{aligned}$$

We will now prove that this is in fact a valid reduction.

$\Rightarrow$: If $I = (\{S_1, ..., S_n\}, U, k)$ is a yes-instance, there exists a set of indices $J \subseteq \{1, ..., n\}$ of at most size k such that $\bigcup_{j \in J} S_j = U$. We construct the weak recursive backdoor $\{s_{j+} | j \in J\}$ of size and depth at most k that dominates all element clauses. Once every variable s_j has been assigned, every element-clause σ_u has been satisfied. Only the $k + 2$ clauses β_i remain. These clauses are disjoint. We can therefore complete the backdoor by adding at depth $k + 1$ every literal b_i simultaneously.

$\Leftarrow$: Let $I = (S, U, k)$ be an instance for the Set Cover, and assume that $I' = (\varphi, k)$ is a yes-instance. Then there must exist a weak recursive backdoor of depth at most k that reduces φ to an edgeless graph. In order to satisfy every β_i clause, each of the b_i literals must be contained in the backdoor. Therefore the size of the backdoor is at least $k + 2$. Since the backdoor can have depth at most $k + 1$, at least two of the β_i clauses have to be split into disconnected components and satisfied separately at some point.

Since every variable b_i is connected to every element-clause σ_u, the graph only contains one connected component, as long as a clause σ_u is not satisfied. Since the literals $\neg b_i$ cannot be part of the backdoor, there must be a set of only set-variables that when assigned satisfy every element-clause. In order to not exceed the recursion depth of $k + 1$, that set must be of size at most k. Having a set of at most k set variables satisfying every element-clause implies the existence of a set cover of U of at most k elements. So I is a yes-instance. ◀