SAT Backdoors: Depth Beats Size

Jan Dreier
Algorithms and Complexity Group, TU Wien, Austria

Sebastian Ordyniak
Algorithms and Complexity Group, University of Leeds, UK

Stefan Szeider
Algorithms and Complexity Group, TU Wien, Austria

Abstract

For several decades, much effort has been put into identifying classes of CNF formulas whose satisfiability can be decided in polynomial time. Classic results are the linear-time tractability of Horn formulas (Aspvall, Plass, and Tarjan, 1979) and Krom (i.e., 2CNF) formulas (Dowling and Gallier, 1984). Backdoors, introduced by Williams, Gomes and Selman (2003), gradually extend such a tractable class to all formulas of bounded distance to the class. Backdoor size provides a natural but rather crude distance measure between a formula and a tractable class. Backdoor depth, introduced by Mählmann, Siebertz, and Vigny (2021), is a more refined distance measure, which admits the utilization of different backdoor variables in parallel. Bounded backdoor size implies bounded backdoor depth, but there are formulas of constant backdoor depth and arbitrarily large backdoor size.

We propose FPT approximation algorithms to compute backdoor depth into the classes Horn and Krom. This leads to a linear-time algorithm for deciding the satisfiability of formulas of bounded backdoor depth into these classes. We base our FPT approximation algorithm on a sophisticated notion of obstructions, extending Mählmann et al.'s obstruction trees in various ways, including the addition of separator obstructions. We develop the algorithm through a new game-theoretic framework that simplifies the reasoning about backdoors.

Finally, we show that bounded backdoor depth captures tractable classes of CNF formulas not captured by any known method.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases satisfiability, backdoor (depth)

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.46

Related Version *Full Version*: `https://arxiv.org/abs/2202.08326`

Funding *Sebastian Ordyniak*: Supported by the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1).
Stefan Szeider: supported by the Austrian Science Fund (FWF, project P32441) and the Vienna Science and Technology Fund (WWTF, project ICT19-065).

1 Introduction

Deciding the satisfiability of a propositional formula in conjunctive normal form (CnfSat) is one of the most important NP-complete problems [4, 16]. Despite its theoretical intractability, heuristic algorithms work surprisingly fast on real-world CnfSat instances [7]. A common explanation for this discrepancy between theoretical hardness and practical feasibility is the presence of a certain "hidden structure" in realistic CnfSat instances [14]. There are various approaches to capturing the vague notion of a "hidden structure" with a mathematical concept. One widely studied approach is to consider the hidden structure in terms of decomposability. For instance, CnfSat can be solved in quadratic time for classes of CNF formulas of bounded branchwidth [2] or bounded treewidth [24].

A complementary approach proposed by Williams et al. [26] considers the hidden structure of a CnfSat instance in terms of a small number of key variables, called *backdoor variables*, that when instantiated move the instance into a polynomial-time solvable class. More precisely, a *backdoor*[1] of *size* k of a CNF formula F into a polynomial-time solvable class $\mathcal{C}$ is a set B of k variables such that for all partial assignments τ to B, the instantiated formula $F[\tau]$ belongs to $\mathcal{C}$. In fact, CnfSat can be solved in linear time for any class of CNF formulas that admit backdoors of *bounded size* into the class of *Horn*, *dual Horn* or *Krom* (i.e., 2CNF) formulas [2].

The size of a smallest backdoor of a CNF formula F into a class $\mathcal{C}$ is a fundamental but rather simple distance measure between F and $\mathcal{C}$. Mählmann, Siebertz, and Vigny [17] proposed to consider instead the smallest *depth* over all backdoors of a formula F into a class $\mathcal{C}$ as distance measure. It is recursively defined as follows:

$$\mathrm{depth}_{\mathcal{C}}(F) := \begin{cases} 0 & \text{if } F \in \mathcal{C} \\ 1 + \min\limits_{x \in var(F)} \max\limits_{\epsilon \in \{0,1\}} \mathrm{depth}_{\mathcal{C}}(F[x=\epsilon]) & \text{if } F \notin \mathcal{C} \text{ and } F \text{ is connected} \\ \max\limits_{F' \in \mathrm{Conn}(F)} \mathrm{depth}_{\mathcal{C}}(F') & \text{otherwise} \end{cases} \quad (1)$$

$\mathrm{Conn}(F)$ denotes the set of connected components of F; precise definitions are given in Section 2. We can certify $\mathrm{depth}_{\mathcal{C}}(F) \leq k$ with a *component $\mathcal{C}$-backdoor tree* of depth $\leq k$ which is a decision tree that reflects the choices made in the above recursive definition.

Backdoor depth is based on the observation that if an instance F decomposes into multiple connected components of $F[x=0]$ and $F[x=1]$, then each component can be treated independently. This way, one is allowed to use in total an unbounded number of backdoor variables. However, as long as the depth of the component $\mathcal{C}$-backdoor tree is bounded, one can still utilize the backdoor variables to solve the instance efficiently. In the context of graphs, similar ideas are used in the study of tree-depth [19, 20] and elimination distance [3, 6]. Bounded backdoor size implies bounded backdoor depth, but there are classes of formulas of unbounded backdoor size but bounded backdoor depth.

The challenging algorithmic problem $\mathcal{C}$-Backdoor Depth is to find for a fixed base class $\mathcal{C}$ and a given formula F, a component $\mathcal{C}$-backdoor tree of F of depth $\leq k$. Mählmann et al. [17] gave an FPT-approximation algorithm for this problem, with k as the parameter) where $\mathcal{C}$ is the trivial class Null for formulas without variables. A component Null-backdoor tree must instantiate all variables of F.

New Results. In this paper, we give the first positive algorithmic results for backdoor depth into nontrivial classes. A minimization problem admits a *standard fixed-parameter tractable approximation (FPT-approximation)* [18] if for an instance of size n and parameter k there is an *FPT-algorithm*, i.e., an algorithm running in time $f(k)n^{\mathcal{O}(1)}$, that either outputs a solution of size at most $g(k)$ or outputs that the instance has no solution of size at most k, for some computable functions f and g; $g(k)$ is also referred to as the performance ratio of the algorithm.

▶ **Main Result 1** (Theorem 15). *$\mathcal{C}$-Backdoor Depth admits an FPT-approximation algorithm if $\mathcal{C}$ is any of the Schaefer classes Horn, dual Horn, or Krom.*

[1] We focus only on strong backdoors, as weak backdoors only apply to satisfiable formulas.
[2] According to Schaefer's Theorem [25], these three classes are the largest nontrivial classes of CNF formulas defined in terms of a property of clauses, for which CnfSat can be solved in polynomial time.

Since our FPT algorithms have linear running time for fixed backdoor depth k, we obtain the following corollary:

▶ **Main Result 2** (Corollary 16). *CnfSat can be solved in linear time for formulas of bounded backdoor depth into the Schaefer classes Horn, dual Horn, and Krom.*

Backdoor depth is a powerful parameter that is able to capture and exploit structure in CnfSat instances that is not captured by any other known method. We list some well-known parameters which render CnfSat fixed-parameter tractable (the list is not complete but covers some of the most essential parameters). For all these parameters, there exist CNF formulas with constant backdoor depth (into *Horn*, *dual Horn*, and *Krom*) but where the other parameter is arbitrarily large: (i) backdoor size into Horn, dual Horn, and Krom [21]; (ii) number of leaves of backdoor trees into Horn, dual Horn, and Krom [23, 22]; (iii) backdoor depth into the class of variable-free formulas [17]; (iv) backdoor treewidth to Horn, dual Horn, and Krom [9, 8]; (v) backdoor size into heterogeneous base classes based on Horn, dual Horn, and Krom [11]; (vi) backdoor size into scattered base classes based on Horn, dual Horn, and Krom [10]; (vii) deletion backdoor size into the class of quadratic Horn formulas [12]; (viii) backdoor size into bounded incidence treewidth [13]. We give definitions and separation proofs in the full version.

Approach and Techniques. A common approach to construct backdoors is to compute in parallel both an upper bound and a lower bound. The upper bounds are obtained by constructing the backdoor itself, and lower bounds are usually obtained in the form of so-called *obstructions*. These are parts of an instance that are proven to be "far away" from the base class. Our results and techniques build upon the pioneering work by Mählmann et al. [17], who introduce *obstruction trees* for backdoor depth. A main drawback of their approach is that it is limited to the trivial base class Null, where the obstructions are rather simple because they can contain only boundedly many variables. Our central technical contribution is overcoming this limitation by introducing *separator obstructions*.

Separator obstructions allow us to algorithmically work with obstruction trees containing an unbounded number of variables, an apparent requirement for dealing with nontrivial base classes different form Null. In the context of backdoor depth, it is crucial that an existing obstruction is disjoint from all potential future obstructions, so they can later be joined safely into a new obstruction of increased depth. Mählmann et al. [17] ensure this by placing the whole current obstruction tree into the backdoor – an approach that only works for the most trivial base class because only there the obstructions have a bounded number of variables. As one considers more and more general base classes, one needs to construct more and more complex obstructions to prove lower bounds. For example, as instances of the base class no longer have bounded diameter (of the incidence graph of the formula) or bounded clause length, neither have the obstructions one needs to consider. Such obstructions become increasingly hard to separate. Our separator obstructions can separate obstruction trees containing an unbounded number of variables from all potential future obstruction trees. We obtain backdoors of bounded depth by combining the strengths of separator obstructions and obstruction trees. We further introduce a *game-theoretic framework* to reason about backdoors of bounded depth. With this notion, we can compute winning strategies instead of explicitly constructing backdoors, greatly simplifying the presentation of our algorithms.

We provide the proofs of statements marked with ⋆ in the full version.

2 Preliminaries

Satisfiability. A *literal* is a propositional variable x or a negated variable $\neg x$. A *clause* is a finite set of literals that does not contain a complementary pair x and $\neg x$ of literals. A propositional formula in conjunctive normal form, or *CNF formula* for short, is a set of clauses. We denote by $\mathcal{CNF}$ the class of all CNF formulas. Let $F \in \mathcal{CNF}$ and $c \in F$. We denote by $var(c)$ the set of all variables occurring in c, i.e., $var(c) = \{ x \mid x \in c \vee \neg x \in c \}$ and we set $var(F) = \bigcup_{c \in F} var(c)$. For a set of literals L, we denote by $\overline{L} = \{ \neg l \mid l \in L \}$, the set of complementary literals of the literals in L. The *size* of a CNF formula F is $\|F\| = \sum_{c \in F} |c|$.

Let $\tau : X \to \{0,1\}$ be an assignment of some set X of propositional variables. If $X = \{x\}$ and $\tau(x) = \epsilon$, we will sometimes also denote the assignment τ by $x = \epsilon$ for brevity. We denote by $\text{true}(\tau)$ ($\text{false}(\tau)$) the set of all literals satisfied (falsified) by τ, i.e., $\text{true}(\tau) = \{ x \in X \mid \tau(x) = 1 \} \cup \{ \neg x \in \overline{X} \mid \tau(x) = 0 \}$ ($\text{false}(\tau) = \overline{\text{true}(\tau)}$). We denote by $F[\tau]$ the formula obtained from F after removing all clauses that are satisfied by τ and from the remaining clauses removing all literals that are falsified by τ, i.e., $F[\tau] = \{ c \setminus \text{false}(\tau) \mid c \in F \text{ and } c \cap \text{true}(\tau) = \emptyset \}$. We say that an assignment satisfies F if $F[\tau] = \emptyset$. We say that F is *satisfiable* if there is some assignment $\tau : var(F) \to \{0,1\}$ that satisfies F, otherwise F is *unsatisfiable*. CnfSat denotes the propositional satisfiability problem, which takes as instance a CNF formula, and asks whether the formula is satisfiable.

The *incidence graph* of a CNF formula F is the bipartite graph G_F whose vertices are the variables and clauses of F, and where a variable x and a clause c are adjacent if and only if $x \in var(c)$. We identify a subgraph G' of the incidence graph G_F with the formula F' consisting of all the clauses of F that are in G', each restricted to the adjacent variables in G'. With slight abuse of notation, we define $var(F')$ to be the variables occuring in G'. Via incidence graphs, graph theoretic concepts directly translate to CNF formulas. For instance, we say that F is *connected* if G_F is connected, and F' is a *connected component* of F if F' is a maximal connected subset of F. $\text{Conn}(F)$ denotes the set of connected components of F. We will also consider the *primal graph* of a CNF formula F, which has as vertex set $var(F)$, and has pairs of variables $x, y \in var(F)$ adjacent if and only of $x, y \in var(c)$ for some $c \in F$.

Base classes. Let $\alpha \subseteq \{+,-\}$ with $\alpha \neq \emptyset$, let $F \in \mathcal{CNF}$ and $c \in F$. We say that a literal l is an *α-literal* if is a positive literal and $+ \in \alpha$ or it is a negative literal and $- \in \alpha$. We say that a variable v *α-occurs* in in a clause c, if v or $\neg v$ is an α-literal that is contained in c. We denote by $var_\alpha(c)$ the set of variables that α-occur in c. For $\alpha \subseteq \{+,-\}$ with $\alpha \neq \emptyset$ and $s \in \mathbb{N}$, let $\mathcal{C}_{\alpha,s}$ be the class of all CNF formulas F such that every clause of F contains at most s α-literals. For $\mathcal{C} \subseteq \mathcal{CNF}$, we say that a clause c is *$\mathcal{C}$-good* if $\{c\} \in \mathcal{C}$. Otherwise, c is *$\mathcal{C}$-bad*. Let τ be any (partial) assignment of the variables of F. We will frequently make use of the fact that $\mathcal{C}_{\alpha,s}$ is *closed under assignments*, i.e., if $F \in \mathcal{C}_{\alpha,s}$, then also $F[\tau] \in \mathcal{C}_{\alpha,s}$. Therefore, whenever a clause $c \in F$ is $\mathcal{C}_{\alpha,s}$-good it will remain $\mathcal{C}_{\alpha,s}$-good in $F[\tau]$ and conversely whenever a clause is $\mathcal{C}_{\alpha,s}$-bad in $F[\tau]$ it is also $\mathcal{C}_{\alpha,s}$-bad in F.

The classes $\mathcal{C}_{\alpha,s}$ capture (according to Schaefer's Dichotomy Theorem [25]) the largest syntactic classes of CNF formulas for which the satisfiability problem can be solved in polynomial time: The class $\mathcal{C}_{\{+\},1} =$ Horn of *Horn formulas*, the class of $\mathcal{C}_{\{-\},1} =$ dHorn of *dual Horn formulas*, and the class $\mathcal{C}_{\{+,-\},2} =$ Krom of *Krom* (or *2CNF*) *formulas*. Note also that the class Null of formulas containing no variables considered by Mählmann et al. [17] is equal to $\mathcal{C}_{\{+,-\},0}$. We follow Williams et al. [26] and focus on classes that are closed under assignments and therefore we do not consider the classes of 0/1-valid and affine formulas.

Note that every class $\mathcal{C}_{\alpha,s}$ (and therefore also the classes of Krom, Horn, and dual Horn formulas) is trivially *linear-time recognizable*, i.e., membership in the class can be tested in linear-time. We say that a class $\mathcal{C}$ of formulas is *tractable* or *linear-time tractable*, if CnfSat restricted to formulas in $\mathcal{C}$ can be solved in polynomial-time or linear-time, respectively. The classes Horn, dHorn, Krom are linear-time tractable [1, 5].

3 Backdoor Depth

A *binary decision tree* is a rooted binary tree T. Every inner node t of T is assigned a propositional variable, denoted by $var(t)$, and has exactly one left and one right child, which corresponds to setting the variable to 0 or 1, respectively. Moreover, every variable occurs at most once on any root-to-leaf path of T. We denote by $var(T)$ the set of all variables assigned to any node of T. Finally, we associate with each node t of T, the truth assignment τ_t that is defined on all the variables $var(P) \setminus \{var(t)\}$ occurring on the unique path P from the root of T to t such that $\tau_t(v) = 0$ ($\tau_t(v) = 1$) if $v \in var(P) \setminus \{var(t)\}$ and P contains the left child (right child) of the node t' on P with $var(t') = v$. Let $\mathcal{C}$ be a base class, F be a CNF formula, and T be a decision tree with $var(T) \subseteq var(F)$. Then T is a *$\mathcal{C}$-backdoor tree of F* if $F[\tau_t] \in \mathcal{C}$ for every leaf t of T [23].

Component backdoor trees generalize backdoor trees as considered by Samer and Szeider [23] by allowing an additional node type, *component nodes*, where the current instance is split into connected components. More precisely, let $\mathcal{C}$ be a base class and F a CNF formula. A *component $\mathcal{C}$-backdoor tree for F* is a pair (T, φ), where T is a rooted tree and φ is a mapping that assigns each node t a CNF formula $\varphi(t)$ such that the following conditions are satisfied:

1. For the root r of T, we have $\varphi(r) = F$.
2. For each leaf ℓ of T, we have $\varphi(\ell) \in \mathcal{C}$.
3. For each non-leaf t of T, there are two possibilities:
   a. t has exactly two children t_0 and t_1, where for some variable $x \in var(\varphi(t))$ we have $\varphi(t_i) = \varphi(t)[x = i]$; in this case we call t a *variable node*.
   b. $\mathrm{Conn}(\varphi(t)) = \{F_1, \ldots, F_k\}$ for $k \geq 2$ and t has exactly k children $t_1, \ldots, t_k$ with $\varphi(t_i) = F_i$; in this case we call t a *component node*.

Thus, a backdoor tree is just a component backdoor tree without component nodes. The *depth* of a $\mathcal{C}$-backdoor is the largest number of variable nodes on any root-to-leaf path. The *$\mathcal{C}$-backdoor depth* $\mathrm{depth}_{\mathcal{C}}(F)$ of a formula F into a base class $\mathcal{C}$ is the smallest depth over all component $\mathcal{C}$-backdoor trees of F. Alternatively, we can define $\mathcal{C}$-backdoor depth recursively as in equation (1) from the introduction. For a component backdoor tree (T, φ) let $var(T, \varphi)$ be the set of all variables x such that some variable node t of T branches on x. We observe that one can use component $\mathcal{C}$-backdoor trees to decide the satisfiability of a formula.

▶ **Lemma 1** ($\star$). *Let $\mathcal{C} \subseteq \mathcal{CNF}$ be tractable, let $F \in \mathcal{CNF}$, and let (T, φ) be a component $\mathcal{C}$-backdoor tree of F of depth d. Then, we can decide the satisfiability of F in time $(2^d \|F\|)^{\mathcal{O}(1)}$. Moreover, if $\mathcal{C}$ is linear-time tractable, then the same can be done in time $\mathcal{O}(2^d \|F\|)$.*

Let $\mathcal{C} \subseteq \mathcal{CNF}$ and $F \in \mathcal{CNF}$. A (strong) *$\mathcal{C}$-backdoor* of F is a set $B \subseteq var(F)$ such that $F[\tau] \in \mathcal{C}$ for each $\tau : B \to \{0, 1\}$. Assume $\mathcal{C}$ is closed under partial assignments (which is the case for many natural base classes and the classes $\mathcal{C}_{\alpha,s}$) and (T, φ) a component $\mathcal{C}$-backdoor tree of F. Then $var(T, \varphi)$ is a $\mathcal{C}$-backdoor of F.

4 Technical Overview

We present all our algorithms in this work within a game-theoretic framework. This framework builds upon the following equivalent formulation of backdoor depth using splitter games. Similar games can be used to describe treedepth and other graph classes [15].

▶ **Definition 2.** *Let $\mathcal{C} \subseteq \mathcal{CNF}$ and $F \in \mathcal{CNF}$. We denote by $\textsc{Game}(F, \mathcal{C})$ the so-called* $\mathcal{C}$-backdoor depth game on F. *The game is played between two players, the* connector *and the* splitter. *The* positions *of the game are CNF formulas. At first, the connector chooses a connected component of F to be the* starting position *of the game. The game is over once a position in the base class $\mathcal{C}$ is reached. We call these positions the* winning positions *(of the splitter). In each* round *the game progresses from a* current position J *to a* next position *as follows:*

- *he splitter chooses a variable $v \in var(J)$.*
- *The connector chooses an assignment $\tau\colon \{v\} \to \{0,1\}$ and a connected component J' of $J[\tau]$. The* next position *is J'.*

In the (unusual) case that a position J contains no variables anymore but J is still not in $\mathcal{C}$, the splitter looses. For a position J, we denote by τ_J the assignment of all variables assigned up to position J.

The following observation follows easily from the definition of the game and the fact that the (strategy) tree obtained by playing all possible plays of the connector against a given d-round winning strategy for the splitter forms a component backdoor tree of depth d, and vice versa. In particular, the splitter choosing a variable v at position J corresponds to a variable node and the subsequent choice of the connector for an assignment τ of v and a component of $J[\tau]$ corresponds to a component node (and a subsequent variable or leaf node) in a component backdoor tree.

▶ **Observation 3.** *The splitter has a strategy for the game $\textsc{Game}(F, \mathcal{C})$ to reach within at most d rounds a winning position if and only if F has $\mathcal{C}$-backdoor depth at most d.*

Using backdoor depth games, we no longer have to explicitly construct a backdoor. Instead, we present so called *splitter-algorithms* that play the backdoor depth game from the perspective of the splitter. The algorithms will have some auxiliary internal state that they modify with each move. Formally, a splitter-algorithm for the $\mathcal{C}$-backdoor depth game to a base class $\mathcal{C}$ is a procedure that

- gets as input a (non-winning) position J of the game, together with an internal state
- and returns a valid move for the splitter at position J, together with an updated internal state.

We will usually use the internal state to hold an obstruction that the splitter will periodically increase in size. Assume we have a game $\textsc{Game}(F, \mathcal{C})$ and some additional input S. For a given strategy of the connector, the splitter-algorithm plays the game as one would expect: In the beginning, the internal state is initialized with S (if no additional input is given, the state is initialized empty). Whenever the splitter should make its next move, the splitter-algorithm is queried using the current position and internal state, and afterwards the internal state is updated accordingly.

▶ **Definition 4.** *We say a splitter-algorithm* implements a strategy to reach for a game $\textsc{Game}(F, \mathcal{C})$ and input S within at most d rounds a position and internal state with some property *if and only if initializing the internal state with S and then playing $\textsc{Game}(F, \mathcal{C})$ according to the splitter-algorithm leads – no matter what strategy the connector is using – after at most d rounds to a position and internal state with said property.*

The following observation converts splitter-algorithms into algorithms for bounded depth backdoors. It builds component backdoor trees by trying all moves of the connector.

▶ **Lemma 5** (⋆). *Let $\mathcal{C} \subseteq \mathcal{CNF}$ and $f_{\mathcal{C}} \colon \mathbb{N} \to \mathbb{N}$. Assume there exists a splitter-algorithm that implements a strategy to reach in each play in the game $\textsc{Game}(F, \mathcal{C})$ and non-negative integer d within at most $f_{\mathcal{C}}(d)$ rounds either:*

i) *a winning position, or*

ii) *(an internal state representing) a proof that the $\mathcal{C}$-backdoor depth of F is at least d.*

Further assume this splitter-algorithm always takes at most $\mathcal{O}(\|F\|)$ time to compute its next move. Then there is an algorithm that, given F and d, in time at most $3^{f_{\mathcal{C}}(d)}\mathcal{O}(\|F\|)$ either:

i) *returns a component $\mathcal{C}$-backdoor tree of depth at most $f_{\mathcal{C}}(d)$, or*

ii) *concludes that the $\mathcal{C}$-backdoor depth of F is at least d.*

For the sake of readability, we may present splitter-algorithms as continuously running algorithms that periodically output moves (via some output channel) and always immediately as a reply get the next move of the connector (via some input channel). Such an algorithm can easily be converted into a procedure that gets as input a position and internal state and outputs a move and a modified internal state: The internal state encodes the whole state of the computation, (e.g., the current state of a Turing machine together with the contents of the tape and the position of the head). Whenever the procedure is called, it "unfreezes" this state, performs the computation until it reaches its next move and then "freezes" and returns its state together with the move.

Our main result is an approximation algorithm (Theorem 15) that either concludes that there is no backdoor of depth d, or computes a component backdoor tree of depth at most $2^{2^{\mathcal{O}(d)}}$. By Lemma 5, this is equivalent to a splitter-algorithm that plays for $2^{2^{\mathcal{O}(d)}}$ rounds to either reach a winning position or a proof that the backdoor depth is larger than d.

Following the approach of Mählmann et al. [17], our proofs of high backdoor depth come in the form of so-called *obstruction trees*. These are trees in the incidence graph of a CNF formula. Their node set therefore consists of both variables and clauses. Obstruction trees of depth d describe parts of an instance for which the splitter needs more than d rounds to win the backdoor depth game. For depth zero, we simply take a single (bad) clause that is not allowed by the base class. Roughly speaking, an obstruction tree of depth $d > 0$ is built from two "separated" obstruction trees T_1, T_2 of depth $d-1$ that are connected by a path. Since the two obstruction trees are separated but in the same component, we know that for any choice of the splitter (i.e., choice of a variable v), there is a response of the connector (i.e., an assignment of v and a component) in which either T_1 or T_2 is whole. Then the splitter needs by induction still more than $d-1$ additional rounds to win the game.

▶ **Definition 6.** *Let $F \in \mathcal{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. We inductively define $\mathcal{C}$-obstruction trees T for F of increasing depth.*

- *Let c be a $\mathcal{C}$-bad clause of F. The set $T = \{c\}$ is a* $\mathcal{C}$-obstruction tree in F of depth 0.
- *Let T_1 be a $\mathcal{C}$-obstruction tree of depth i in F. Let β be a partial assignment of the variables in F. Let T_2 be an obstruction tree of depth i in $F[\beta]$ such that no variable $v \in var(F[\beta])$ occurs both in a clause of T_1 and T_2. Let further P be (a CNF formula representing) a path that connects T_1 and T_2 in F. Then $T = T_1 \cup T_2 \cup var(P) \cup P$ is a* $\mathcal{C}$-obstruction tree in F of depth $i+1$.

▶ **Lemma 7** (⋆). *Let $F \in \mathcal{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. If there is a $\mathcal{C}$-obstruction tree of depth d in F, then the $\mathcal{C}$-backdoor depth of F is larger than d.*

Our splitter-algorithm will construct obstruction trees of increasing depth by a recursive procedure (Lemma 14) that we outline now. We say a splitter-algorithm satisfies *property* i if it reaches in each game $\textsc{Game}(F, \mathcal{C})$ within $g_{\mathcal{C}}(i, d)$ rounds (for some function $g_{\mathcal{C}}(i, d)$) either

1) a winning position, or

2) a position J and a $\mathcal{C}$-obstruction tree T of depth i in F such that no variable in $var(J)$ occurs in a clause of T, or

3) a proof that the $\mathcal{C}$-backdoor depth of F is at least d.

If we have a splitter algorithm satisfying property $d+1$ then our main result, the approximation algorithm for backdoor depth, directly follows from Lemma 7 and Lemma 5. Assume we have a strategy satisfying property $i-1$, let us describe how to use it to satisfy property i. If at any point we reach a winning position, or a proof that the $\mathcal{C}$-backdoor depth of F is at least d, we are done. Let us assume this does not happen, so we can focus on the much more interesting second case.

We use property $i-1$ to construct a first tree T_1 of depth $i-1$, and reach a position J_1. We use it again, starting at position J_1 to construct a second tree T_2 of depth $i-1$ that is completely contained in position J_1. Since in the beginning the connector selected a connected component, T_1 and T_2 are in the same component of F and we can find a path P connecting them. Let β be the assignment that assigns all the variables the splitter chose until reaching position J_1. Then T_2 is an obstruction tree not only in J_1 but also in $F[\beta]$. In order to join both trees together into an obstruction of depth i, we have to show, according to Definition 6 that no variable $v \in var(F[\beta])$ occurs both in a clause of T_1 and T_2. Since no variable in $var(J_1)$ occurs in a clause of T_1 (property $i-1$), and T_2 was built only from J_1, this is the case. The trees T_1 and T_2 are "separated" and can be safely joined into a new obstruction tree T of depth i (see also Figure 3 on page 15 and the proof of Lemma 14 for details).

The last thing we need to ensure is that we reach a position J such that no variable in $var(J)$ occurs in a clause of T. This then guarantees that T is "separated" from all future obstruction trees that we may want to join it with to satisfy property $i+1$, $i+2$ and so forth. This is the major difficulty and main technical contribution of this paper.

It is important to note here, that the exact notion of "separation" between obstruction trees plays a crucial role for our approach and is one of the main differences to Mählmann et al. [17]. Mählmann et al. solve the separation problem in a "brute-force" manner: If we translate their approach to the language of splitter-algorithms, then the splitter simply selects all variables that occur in a clause of T. For their base class – the class Null of formulas without variables – there are at most $2^{\mathcal{O}(d)}$ variables that occur in an obstruction tree of depth d. Thus, in only $2^{\mathcal{O}(d)}$ rounds, the splitter can select all of them, fulfilling the separation property. This completes the proof for the base class Null.

However, already for backdoor depth to Krom, this approach cannot work since instances in the base class have obstruction trees with arbitrarily many clauses. Moreover, the situation becomes even more difficult for backdoors to Horn, since additionally clauses are allowed to contain arbitrary many literals. Mählmann et al. acknowledge this as a central problem and ask for an alternative approach to the separation problem that works for more general base classes.

5 Separator Obstructions

The main technical contribution of this work is a separation technique that works for the base classes $\mathcal{C} = \mathcal{C}_{\alpha,s}$. The separation technique is based on a novel form of obstruction, which we call *separator obstruction*. Obstruction trees are made up of paths, therefore, it is

sufficient to separate each new path P that is added to an obstruction. Note that P can be arbitrarily long and every clause on P can have arbitrary many variables and therefore the splitter cannot simply select all variables in (clauses of) P. Instead, given such a path P that we want to separate, we will use separator obstructions to develop a splitter-algorithm (Lemma 12) that reaches in each game $\textsc{Game}(F, \mathcal{C})$ within a bounded number of rounds either

1) a winning position, or
2) a position J such that no variable in $var(J)$ occurs in a clause of P, or
3) a proof that the $\mathcal{C}$-backdoor depth of F is at least d.

Informally, a separator obstruction is a sequence $\langle P_1, \dots, P_\ell \rangle$ of paths that form a tree T_ℓ together with an assignment τ of certain *important* variables occurring in T_ℓ. The variables of τ correspond to the variables chosen by the splitter-algorithm and the assignment τ corresponds to the assignment chosen by the connector. Each path P_i adds at least one $\mathcal{C}$-bad clause b_i to the separator obstruction, which is an important prerequisite to increase the backdoor depth by growing the obstruction. Moreover, by choosing the important variables and the paths carefully, we ensure that for every *outside* variable, i.e., any variable that is not an important variable assigned by τ, there is an assignment and a component (which can be chosen by the connector) that leaves a large enough part of the separator obstruction intact. Thus, if a separator obstruction is sufficiently large, the connector can play such that even after d rounds a non-empty part of the separator obstruction is still intact. This means a large separator obstruction is a proof that the backdoor depth is larger than d.

To illustrate the growth of a separator obstruction (and motivate its definition) suppose that our splitter-algorithm is at position J of the game $\textsc{Game}(F, \mathcal{C})$ and has already built a separator obstruction $X = \langle \langle P_1, \dots, P_i \rangle, \tau \rangle$ (with corresponding tree T_i) containing $\mathcal{C}$-bad clauses $b_1, \dots, b_i$; note that τ is compatible with τ_J (i.e., τ and τ_J agree on the common assigned variables). If J is already a winning position, then property i is satisfied. Therefore, J has to contain a $\mathcal{C}$-bad clause. If no $\mathcal{C}$-bad clause has a path to T_i in J, then J satisfies 2) of property i and we are also done. Otherwise, let b_{i+1} be a $\mathcal{C}$-bad clause in J that is closest to T_i and let P_{i+1} be a shortest path from b_{i+1} to T_i in J. Then, we extend our separator obstruction X by attaching the path P_{i+1} to T_i (and obtain the tree T_{i+1}). Our next order of business is to choose a bounded number of important variables occurring on P_{i+1} that we will add to X. Those variables need to be chosen such that no outside variable can destroy too much of the separator obstruction. Apart from destroying the paths of the separator obstruction, we also need to avoid that assigning any outside variable makes too many of the $\mathcal{C}$-bad clauses $b_1, \dots, b_{i+1}$ $\mathcal{C}$-good. Therefore, a natural choice would be to add all variables of b_{i+1} to X, i.e., to make those variables important. Unfortunately, this is not possible since b_{i+1} can contain arbitrarily many literals. Instead, we will only add the variables of b_{i+1} to X that α-occur in b_{i+1}. By the following lemma, the number of those variables is bounded.

▶ **Lemma 8** ($\star$). *Let $F \in \mathcal{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. If F has $\mathcal{C}$-backdoor depth at most some integer d, then every clause of F contains at most $d + s$ α-literals.*

While this still allows for outside variables to occur in many of the $\mathcal{C}$-bad clauses $b_1, \dots, b_{i+1}$, it already ensures that no outside variable can α-occur in any of these clauses. This helps us, since when $|\alpha| = 1$ (i.e., the only case where α-occurs means something different then just occurs), it provides us with an assignment of any such outside variable that the connector can play without making the $\mathcal{C}$-bad clauses in which it occurs $\mathcal{C}$-good. For instance, if $\alpha = \{+\}$, then any outside variable v can only occur negatively in a $\mathcal{C}$-bad clause and moreover setting v to 0 ensures that the $\mathcal{C}$-bad clauses remain $\mathcal{C}$-bad.

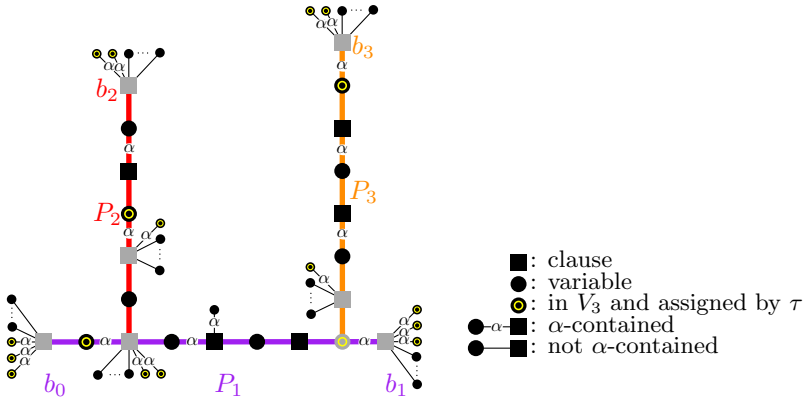

Figure 1 A separator obstruction containing three paths P_1, P_2, and P_3. The figure shows vertices and edges of the incidence graph. Only the colorful edges are part of separator obstruction's tree. Gray variables and clauses are mentioned under the names b_i, e, and c in Definition 9.

Next, we need to ensure that any outside variable cannot destroy too many paths. By choosing a *shortest* path P_{i+1}, we have already ensured that no variable occurs on more than two clauses of P_{i+1} (such a variable would be a shortcut, meaning P_{i+1} was not a shortest path). Moreover, because P_{i+1} is a shortest path from b_{i+1} to T_i, every variable that occurs on T_i and on P_{i+1} must occur in the clause c in P_{i+1} that is closest to T_i but not in T_i itself. Similarly, to how we dealt with the $\mathcal{C}$-bad clauses, we will now add all variables that α-occur in c to X. This ensures that no outside variable can α-occur in both T_i and P_{i+1} , which (by induction over i) implies that every outside variable α-occurs in at most two clauses (either from T_i or from P_{i+1}) and therefore provides us with an assignment for the outside variables that removes at most two clauses from X. However, since removing any single clause can be arbitrarily bad if the clause has a high degree in the separator obstruction, we further need to ensure that all clauses of the separator obstruction in which outside variables α-occur have small degree. We achieve this by adding the variables α-occurring in any clause as soon as its degree (in the separator obstruction) becomes larger than two, which happens whenever the endpoint of P_{i+1} in T_i is a clause. Finally, if the endpoint of P_{i+1} in T_i is a variable, we also add this variable to the separator obstruction to ensure that no variable has degree larger than three in T_{i+1}. This leads us to the following definition of separator obstructions (see also Figure 1 for an illustration).

▶ **Definition 9.** *Let $F \in \mathcal{CNF}$ and $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. A* $\mathcal{C}$-separator obstruction *for F is a tuple $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ (where $P_1, \ldots, P_\ell$ are paths in F and τ is an assignment of variables of F) satisfying the following recursive definition.*

- *P_1 is a shortest path between two $\mathcal{C}$-bad clauses b_0 and b_1 in F. Let $B_1 = \{b_0, b_1\}$, let V_1 be the set of all variables that α-occur in any clause in B_1, let $\tau_1 : V_1 \to \{0,1\}$ be any assignment of the variables in V_1, and let $T_1 = P_1$.*
- *For every i with $1 < i \leq \ell$, let b_i be a $\mathcal{C}$-bad clause in $F[\tau_{i-1}]$ of minimal distance to T_{i-1} in $F[\tau_{i-1}]$. Then, P_i is a shortest path (of possibly length zero) in $F[\tau_{i-1}]$ from T_{i-1} to b_i and $T_i = T_{i-1} \cup P_i$. Moreover, let e be the variable or clause that is both in T_{i-1} and P_i. We define B_i and V_i by initially setting $B_i = B_{i-1} \cup \{b_i\}$ and $V_i = V_{i-1} \cup var_\alpha(b_i)$ and distinguishing two cases:*
  - *If e is a variable, then let c be the clause on P_i incident with e (note that it is possible that $c = b_i$). Then, we add c to B_i and we add $\{e\} \cup var_\alpha(c)$ to V_i.*

- *If e is a clause, then either $e = b_i$ or $e \neq b_i$ and there is a clause c that is closest to e on P_i (it may be that $c = b_i$). In the former case we leave B_i and V_i unchanged and in the latter case, we add e and c to B_i and we add $var_\alpha(e) \cup var_\alpha(c)$ to V_i.*

$\tau_i : V_i \to \{0,1\}$ *is any assignment of the variables in V_i that is compatible with τ_{i-1}.*

We set $\tau = \tau_\ell$. The size *of X is the number of paths in $T = T_\ell$, i.e., $\ell + 1$.*

The assignment τ is a central part of the definition, guiding the connector in Lemma 11 and thereby establishing a lower bound on the backdoor depth. We start by observing some simple but important properties of separator obstructions.

▶ **Lemma 10** (⋆). *Let $F \in \mathcal{CNF}$, $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$, and let $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ be a $\mathcal{C}$-separator obstruction in F, then for every $i \in [\ell]$:*

(C1) *T_i is a tree.*

(C2) *Every variable $v \notin V_i$ occurs in at most two clauses of P_j for every j with $1 \leq j \leq i$ and moreover those clauses are consecutive in P_j.*

(C3) *Every variable $v \notin V_i$ α-occurs in at most two clauses of T_i and moreover those clauses are consecutively contained in one path of T_i.*

(C4) *Every variable $v \in V_i \setminus V_{i-1}$ α-occurs in most four clauses of T_i.*

(C5) *If a variable $v \notin V_i$ α-occurs in a clause c of T_i, then c has degree at most two in T_i.*

(C6) *Every variable of F has degree at most three in T.*

(C7) *If every clause of F contains at most x α-literals, then $|V_i \setminus V_{i-1}| \leq 2s + x + 1$.*

We now show the main result of this subsection, namely, that also separator obstructions can be used to obtain a lower bound on the backdoor depth of CNF formulas.

▶ **Lemma 11.** *Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$ and $F \in \mathcal{CNF}$. If F has a $\mathcal{C}$-separator obstruction of size at least $\ell = (8^d(14^2 + 2d))^{2^d}$, then F has $\mathcal{C}$-backdoor depth at least d.*

Proof. Let $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ be a $\mathcal{C}$-separator obstruction for F of size at least ℓ with V_i, B_i, T_i, T as in Definition 9. Let J be a position in the game $\textsc{Game}(F, \mathcal{C})$. We say that a subtree T' of $T = T_\ell$ is *contained* in J if every variable and clause of T' occurs in J. Let T' be a subtree of T that is contained in J. Let P_j be a path of X. We say that P_j is *active* in T' if either $V(P_j) = \{b_j\}$ and T' contains b_j or T' contains a vertex in $V(P_j) \setminus V(T_{j-1})$. Moreover, we say that P_j is *intact* in T' at position J if $V(P_j) \subseteq V(T')$ and b_j is a $\mathcal{C}$-bad clause in J. Otherwise, we say that P_j is *broken* in T' at position J.

We show by induction on the number of rounds that there is a strategy S for the connector such that the following holds for every position J reached after i rounds in the game $\textsc{Game}(F, \mathcal{C})$ against S: At position J, there is a subtree T' of T contained in J that contains at least $\ell_i = (\ell^{(1/2)^i}/8^i)$ intact paths and at most $z_i = 2i$ broken paths of X. This then shows the statement of the lemma because $\ell_d = \ell^{1/2^d}/8^d = 14^2 + 2d \geq 1$ and therefore any position J reached after d rounds in the game $\textsc{Game}(F, \mathcal{C})$ contains at least one clause that is $\mathcal{C}$-bad in J.

The claim clearly holds for $i = 0$ since $\ell_0 = \ell$ and $z_0 = 0$ and the connector can choose the component of F containing T. Assume now that $i > 0$ and let J be the position reached after $i - 1$ rounds. By the induction hypothesis, at position J there is a subtree T' of T contained in J containing at least $\ell_{i-1} = \ell^{(1/2)^{i-1}}/8^{i-1}$ intact paths and at most $z_{i-1} = 2(i-1)$ broken paths of X. Suppose that the splitter chooses variable v as its next move. Moreover, let o be the smallest integer such that $v \in V_o$; if $v \notin V_\ell$ we set $o = \ell + 1$. Note that $v \notin V_j$ for every

$i < o$. Let I be the set of all paths P_j of X that are intact in T' at position J and let $I_{<o}$ ($I_{>o}$) be the subset of I containing only the paths P_j with $j < o$ ($j > o$). Finally, let $T'_{<o}$ be the subtree of T' restricted to the paths P_j of X with $j < o$. Note that at position J, $T'_{<o}$ is connected and the paths in $I_{<o}$ are intact also in $T'_{<o}$. Then, the connector chooses the assignment $\beta : \{v\} \to \{0, 1\}$ such that:

$$\beta(v) = \begin{cases} \tau(v) & |I_{<o}| < \sqrt{\ell_{i-1}}, \\ 1 & |I_{<o}| \geq \sqrt{\ell_{i-1}} \text{ and } + \in \alpha, \\ 0 & \text{otherwise.} \end{cases}$$

As we will show below, β is defined in such a manner that the position $J' = J[\beta]$ reached after the next round of the game $\textsc{Game}(F, \mathcal{C})$ contains a subtree T'' of T' containing at least $\ell_i = \sqrt{\ell_{i-1}}/8$ paths that are intact in J' and at most $z_i = z_{i-1} + 2$ broken paths, which completes the proof since the connector can now chose the component of J' containing T'' to fulfill the induction invariant. We distinguish the following cases; refer also to Figure 2 for an illustration of the two cases.

Case 1: $|I_{<o}| \geq \sqrt{\ell_{i-1}}$. We will show that T'' can be obtained as a subtree of $T'_{<o}$.

Note first that all clauses b_j with $j < o$ that are $\mathcal{C}$-bad in J are also $\mathcal{C}$-bad in J'. This is because $v \notin V_j$ (because $j < o$ and $v \notin V_{o-1}$) and therefore v cannot α-occur in b_j, which implies that b_j remains $\mathcal{C}$-bad and not satisfied after setting v to $\beta(v)$.

The tree $T'_{<o}$ in J may decompose into multiple components in J'. We will argue that one of these components contains many intact paths and only at most two more broken paths than $T'_{<o}$. Since the $\mathcal{C}$-bad clauses of an intact path remain $\mathcal{C}$-bad in J', the only way in which an intact path can become broken is if parts of the path get removed, i.e., either v or clauses satisfied by setting v to $\beta(v)$.

If $\beta(v) = 1$ then $+ \in \alpha$. If $\beta(v) = 0$ then $+ \notin \alpha$, and since $\alpha \neq \emptyset$, then $- \in \alpha$. Thus, in $J' = J[\beta]$, the only elements that are removed are the variable v as well as clauses in which v α-occurs. By Lemma 10 (C3), v α-occurs in at most two clauses of $T'_{<o}$ and because of (C5) those clauses have degree at most two in $T'_{<o}$. Therefore, setting v to $\beta(v)$ removes at most two clauses from $T'_{<o}$, each of which having degree at most two. Moreover, according to Lemma 10 (C6), v itself has degree at most three in $T'_{<o}$. This implies that setting v to $\beta(v)$ splits $T'_{<o}$ into at most $2 \cdot 2 + 3 = 7$ components.

Moreover, because of Lemma 10 (C3), the at most two clauses of $T'_{<o}$ in which v α-occurs are located on the same path P_j. Therefore, at most two paths that are complete in $T'_{<o}$, i.e., the path P_j and the at most one path containing v, can become broken. Therefore, there is a component of J' that contains a subtree of $T'_{<o}$ that contains at least $|I_{<o}|/7 - 2 \geq \sqrt{\ell_{i-1}}/7 - 2$ intact paths and at most $z_{i-1} + 2 \leq 2i = z_i$ broken paths of X. Note that $\sqrt{\ell_{i-1}} \geq \ell_d \geq 14^2 + 2d \geq 14^2$ and therefore $\sqrt{\ell_{i-1}}/7 - 2 \geq \sqrt{\ell_{i-1}}/8 = \ell_i$.

Case 2: $|I_{<o}| < \sqrt{\ell_{i-1}}$. This means $\beta(v) = \tau(v)$. In this case, we will build the subtree T'' by picking only one path from $T'_{<o}$ and the remaining paths from $P_{o+1}, \ldots, P_\ell$. Let A be the set of all paths of X that are active in T' and let $A_{>o}$ ($A_{<o}$) be the subset of A containing only the paths P_j with $j > o$ ($j < o$). We say that a path P_a of X is *attached* to a path P_b of X if $a > b$, $V(P_a) \cap V(P_b) \neq \emptyset$ and there is no $b' < b$ with $V(P_a) \cap V(P_{b'}) \neq \emptyset$. We say that a path P_a in $A_{>o}$ is *weakly attached* to a path P_b in $A_{<o}$ if either:

- P_a is attached to P_b or
- P_a is attached to a path P_c in $A_{>o}$ that is in turn weakly attached to P_b.

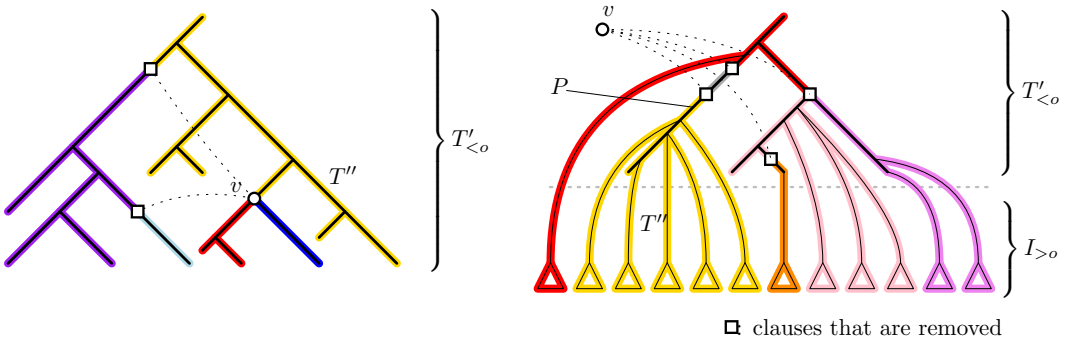

Figure 2 Left: Case 1. The set $I_{<o}$ is large. Assigning v to $\beta(v)$ decomposes the tree $T'_{<o}$ into at most seven components. The largest component T'' is still large. Right: Case 2. The set $I_{<o}$ is small. There is a path P to which many paths are weakly attached, forming a tree T_P. Assigning v to $\beta(v)$ splits T_P in at most three parts. The largest component T'' of T_P is still large.

Note that because T' is a tree, every path in $A_{>o}$ is weakly attached to exactly one path in $A_{<o}$. Moreover, for the same reason any path in $A_{<o}$ together with all paths in $A_{>o}$ that are weakly attached to it forms a subtree of T'.

Therefore, there is a path P in $A_{<o}$ such that at least $|I_{>o}|/|A_{<o}|$ paths in $I_{>o}$ are weakly attached to P. Moreover, the union T_P of P and all paths in $A_{>o}$ that are weakly attached to P is a subtree of T'. Note that T_P has at least $|I_{>o}|/|A_{<o}|$ paths that are intact in T_P and at most z_{i-1} paths that are broken in T_P at position J. Since $\sqrt{\ell_{i-1}} \geq \ell_d = 14^2 + 2d \geq 2d$ and $z_{i-1} \leq z_d = 2d$, it holds that $|I_{<o}| + z_{i-1} \leq 2\sqrt{\ell_{i-1}}$ (because also $|I_{<o}| \leq \sqrt{\ell_{i-1}}$). Therefore,

$$\begin{aligned} |I_{>o}|/|A_{<o}| &\geq (\ell_{i-1} - |I_{<o}|)/(|I_{<o}| + z_{i-1}) \\ &\geq (\ell_{i-1} - |I_{<o}|)/(2|I_{<o}|) \\ &\geq (\ell_{i-1})/(2\sqrt{\ell_{i-1}}) - 1/2 \\ &\geq \sqrt{\ell_{i-1}}/2 - 1/2 \\ &= 8\ell_i/2 - 1 \geq 3\ell_i. \end{aligned}$$

Because $\beta(v) = \tau(v)$, all paths P_j with $j > o$ that are active in T_P are still contained in J' and moreover if P_j is intact in J, then it is still intact in J'. Moreover, because of Lemma 10 (C2), v occurs in at most two clauses of P and because $\beta(v) = \tau(v)$ all paths $P_{o+1}, \cdots, P_\ell$ that are attached to P are still attached to P after setting v to $\beta(v)$. It follows that setting v to $\beta(v)$ removes at most two clauses and at most one variable (i.e., the variable v) from P and also from T_P. Therefore, $J' = J[\beta]$ contains a component that contains a subtree T'' of T_P with at least $3\ell_i/3 = \ell_i$ paths that are intact in T'' and at most $z_{i-1} + 1 \leq z_i$ paths that are broken in T''. ◀

6 Winning Strategies and Algorithms

We are ready to present our algorithmic results. Earlier, we discussed that separator obstructions are used to separate existing obstruction trees from future obstruction trees. As all obstruction trees are built only from shortest paths, it is sufficient to derive a splitter-algorithm that takes a shortest path P and separates it from all future obstructions. By reaching a position J such that no variable in $var(J)$ occurs in a clause of P, we are guaranteed that all future obstructions are separated from P, as future obstructions will only contain clauses and variables from J.

▶ **Lemma 12** (⋆). *Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. There exists a splitter-algorithm that implements a strategy to reach for each game $\textsc{Game}(F,\mathcal{C})$, non-negative integer d, and shortest path P between two $\mathcal{C}$-bad clauses in F within at most $(3s+d+1)(8^d(14^2+2d))^{2^d}$ rounds either:*
1) *a winning position, or*
2) *a position J such that no variable in $var(J)$ occurs in a clause of P, or*
3) *a proof that the $\mathcal{C}$-backdoor depth of F is at least d.*
This algorithm takes at most $\mathcal{O}(\|F\|)$ time per move.

Since selecting more variables can only help the splitter in archiving their goal, we immediately also get the following statement from Lemma 12.

▶ **Corollary 13.** *Consider $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$, a game $\textsc{Game}(F,\mathcal{C})$ and a position J' in this game, a non-negative integer d and shortest path P between two $\mathcal{C}$-bad clauses in F. There exists a splitter-algorithm that implements a strategy that continues the game from position J' and reaches within at most $(3s+d+1)(8^d(14^2+2d))^{2^d}$ rounds either:*
1) *a winning position, or*
2) *a position J such that no variable in $var(J)$ occurs in a clause of P, or*
3) *a proof that the $\mathcal{C}$-backdoor depth of F is at least d.*
This algorithm takes at most $\mathcal{O}(\|F\|)$ time per move.

As described at the end of Section 4, we can now construct in the following lemma obstruction trees of growing size, using the previous corollary to separate them from potential future obstruction trees.

▶ **Lemma 14.** *Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. There is a splitter-algorithm that implements a strategy to reach for a game $\textsc{Game}(F,\mathcal{C})$ and non-negative integers i, d with $1 \leq i \leq d$ within at most $(2^i-1)(3s+d+1)(8^d(14^2+2d))^{2^d}$ rounds either:*
1) *a winning position, or*
2) *a position J and a $\mathcal{C}$-obstruction tree T of depth i in F such that no variable in $var(J)$ occurs in a clause of T, or*
3) *a proof that the $\mathcal{C}$-backdoor depth of F is at least d.*
This algorithm takes at most $\mathcal{O}(\|F\|)$ time per move.

Proof. We will prove this lemma by induction over i. Our splitter-algorithm will try construct an obstruction tree of depth i by first using the induction hypothesis to build two obstruction trees T_1 and T_2 of depth $i-1$ and then joining them together. After the construction of the first tree T_1, we reach a position J_1 and by our induction hypothesis no variable in $var(J_1)$ occurs in a clause of T_1. This encapsulates the core idea behind our approach, as it means that T_1 is separated from all potential future obstruction trees T_2 that we build from position J_1. Therefore, we can compute the next tree T_2 in J_1 and join T_1 and T_2 together in accordance with Definition 6 by a path P. At last, we use Corollary 13 to also separate this path from all future obstructions. If at any point of this process we reach a winning position or a proof that the $\mathcal{C}$-backdoor depth of F is at least d, we can stop. Let us now describe this approach in detail.

For convenience, let $x = (3s+d+1)(8^d(14^2+2d))^{2^d}$. We start our induction with $i=1$. If there is no $\mathcal{C}$-bad clause in F, then it is a winning position and we can stop. Assume there is exactly one $\mathcal{C}$-bad clause c in F. By Lemma 8, if c contains more then $d+s$ α-literals, we have a proof that the $\mathcal{C}$-backdoor depth of F is at least d and we archive case 3) of the lemma. On the other hand, if c contains at most $d+s$ α-literals, the splitter can obtain a

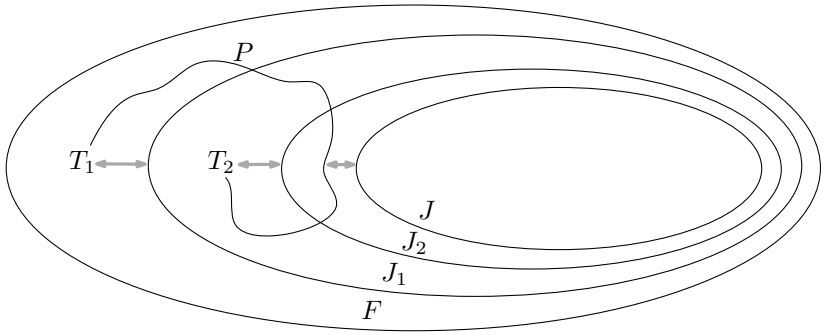

Figure 3 Overview of the construction in Lemma 14. First, T_1 is chosen in F, yielding J_1. Then, T_2 is chosen in J_1, yielding J_2. In the end the connecting path P is chosen yielding J. A gray doublesided arrow between a position $\widehat{J}$ and structure $\widehat{T}$ symbolizes that no variable $v \in var(\widehat{J})$ occurs in a clause of $\widehat{T}$.

winning position in $\textsc{Game}(F, \mathcal{C})$ after at most $d + s \leq (2^i - 1)x$ rounds by choosing a new variable α-occurring in c at every round. Assume there is more than one $\mathcal{C}$-bad clause in F. Thus, we pick $\mathcal{C}$-bad clauses c_1 and c_2 and compute a shortest path P between c_1 and c_2 in F. By Definition 6, $T = \{c_1\} \cup \{c_2\} \cup var(P) \cup P$ is a $\mathcal{C}$-obstruction tree of depth 1 in F. We then continue the game using Corollary 13 (for the path P) to reach a position J' satisfying (1), (2), or (3) after at most $x \leq (2^i - 1)x$ rounds, with each round taking at most $\mathcal{O}(\|F\|)$ time.

We now assume the statement of this lemma to hold for $i - 1$ and we show it also holds for i. To this end, we start playing the game $\textsc{Game}(F, \mathcal{C})$ according to the existing splitter-algorithm for $i-1$. If we reach (within at most $(2^{i-1}-1)x$ rounds) a winning position or a proof that the $\mathcal{C}$-backdoor depth of F is at least d then we are done. Assuming this is not the case, we reach a position J_1 and a $\mathcal{C}$-obstruction tree T_1 of depth $i - 1$ in F such that no variable $v \in var(J_1)$ occurs in a clause of T_1.

We continue playing the game at position J_1 according to the existing splitter-algorithm for $\textsc{Game}(J_1, \mathcal{C})$ and $i - 1$. The $\mathcal{C}$-backdoor depth of F is larger or equal to the $\mathcal{C}$-backdoor depth of J_1. Thus again (after at most $(2^{i-1} - 1)x$ rounds) we either are done (because we reach a winning position or can conclude that the $\mathcal{C}$-backdoor depth of J_1 is at least d) or we reach a position J_2 and a $\mathcal{C}$-obstruction tree T_2 of depth $i - 1$ in J_1 such that no variable $v \in var(J_2)$ occurs in a clause of T_2.

We pick two clauses $c_1 \in T_1$ and $c_2 \in T_2$ that are $\mathcal{C}$-bad in F and compute a shortest path P between c_1 and c_2 in F. We now argue that $T = T_1 \cup T_2 \cup var(P) \cup P$ is a $\mathcal{C}$-obstruction tree of depth i in F. Let $\beta = \tau_{J_1}$ be the assignment that assigns all the variables the splitter chose until reaching position J_1 to the value given by the connector. Note that J_1 is a connected component of $F[\beta]$.

Since all variables and clauses belonging to T_2 induce a connected subgraph of J_1, T_2 is a $\mathcal{C}$-obstruction tree of depth $i - 1$ not only in J_1, but also in $F[\beta]$. Let $v \in var(F[\beta])$. We show that v does not occur both in some clause of T_1 and of T_2. To this end, assume v is contained in a clause of T_2. Since all clauses of T_2 are in J_1 and J_1 is a connected component of $F[\beta]$, we further have $v \in var(J_1)$. On the other hand (as discussed earlier), no variable $v \in var(J_1)$ is contained in a clause of T_1. By Definition 6, $T = T_1 \cup T_2 \cup var(P) \cup P$ is a $\mathcal{C}$-obstruction tree of depth i in F.

We use Corollary 13 to continue playing the game at position J_2. Again, if we reach a winning position or a proof that the $\mathcal{C}$-backdoor depth of F is at least d we are done. So we focus on the third case that we reach (within at most x rounds) a position J such that

no variable $v \in var(J)$ is contained in a clause of P. We know already that no variable $v \in var(J_1)$ is contained in a clause of T_1 and no variable $v \in var(J_2)$ is contained in a clause of T_2. Since $var(J) \subseteq var(J_2) \subseteq var(J_1)$, and $T = T_1 \cup T_2 \cup var(P) \cup P$, we can conclude that no variable $v \in var(J)$ is contained in a clause of T.

In total, we played for $(2^{i-1}-1)x + (2^{i-1}-1)x + x = (2^i - 1)x$ rounds. The splitter-algorithm in Corollary 13 takes at most $\mathcal{O}(\|F\|)$ time per move. The same holds for the splitter-algorithm for $i-1$ that we use as a subroutine. Thus, the whole algorithm takes at most $\mathcal{O}(\|F\|)$ time per move. ◀

The main results now follow easily by combining Lemmas 1, 5, 7, and 14.

▶ **Theorem 15** (⋆). *Let $\mathcal{C} = \mathcal{C}_{\alpha,s}$ with $\alpha \subseteq \{+,-\}$, $\alpha \neq \emptyset$, and $s \in \mathbb{N}$. We can, for a given $F \in \mathcal{CNF}$ and a non-negative integer d, in time at most $2^{2^{2^{\mathcal{O}(d)}}}\|F\|$ either*
1) *compute a component $\mathcal{C}$-backdoor tree of F of depth at most $2^{2^{\mathcal{O}(d)}}$, or*
2) *conclude that the $\mathcal{C}$-backdoor depth of F is larger than d.*

▶ **Corollary 16.** *Let $\mathcal{C} \in \{\text{Horn}, \text{dHorn}, \text{Krom}\}$. The CnfSat problem can be solved in linear time for any class of formulas of bounded $\mathcal{C}$-backdoor depth.*

7 Conclusion

We show that CnfSat can be solved in linear-time for formulas of bounded $\mathcal{C}$-backdoor depth whenever $\mathcal{C}$ is any of the well-known Schaefer classes. We achieve this by showing that $\mathcal{C}$-backdoor depth can be FPT-approximated for any class $\mathcal{C} = \mathcal{C}_{\alpha,s}$. This allows us to extend the results of Mählmann et al. [17] for the class of variable-free formulas to all Schaefer classes. Our results provide an important milestone towards generalizing and unifying the various tractability results based on variants of $\mathcal{C}$-backdoor size (see also future work below) to the only recently introduced and significantly more powerful $\mathcal{C}$-backdoor depth.

Let us finish with some natural and potentially significant extensions of backdoor depth that can benefit from our approach based on separator obstructions. Two of the probably most promising ones that have already been successfully employed as extensions of backdoor size are the so-called *scattered* and *heterogeneous* backdoor sets [11, 10].

Interestingly, while those two notions lead to orthogonal tractable classes in the context of backdoor size, they lead to the same tractable class for backdoor depth. Therefore, lifting these two extensions to backdoor depth, would result in a unified and significantly more general approach. While we are hopeful that our techniques can be adapted to this setting, one of the main remaining obstacles is that obstructions of depth 0 no longer are single (bad) clauses. For instance, consider the heterogeneous class $\mathcal{C} = \text{Horn} \cup \text{Krom}$. Here, a CNF formula may not be in $\mathcal{C}$ due to a pair of clauses, one in $\text{Horn} \setminus \text{Krom}$ and another one in $\text{Krom} \setminus \text{Horn}$. Finally, an even more general but also more challenging tractable class to consider for backdoor depth is the class of Q-Horn formulas, which generalizes the heterogeneous class obtained as the union of all considered Schaefer classes.

References

1 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

2 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 340–351, 2003.

3 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016.

4 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual Symp. on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.

5 William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 1(3):267–284, 1984.

6 Fedor V Fomin, Petr A Golovach, and Dimitrios M Thilikos. Parameterized complexity of elimination distance to first-order logic properties. *arXiv preprint arXiv:2104.02998*, 2021.

7 Vijay Ganesh and Moshe Y. Vardi. On the unreasonable effectiveness of SAT solvers. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 547–566. Cambridge University Press, 2020. doi:10.1017/9781108637435.032.

8 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Backdoor treewidth for SAT. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT 2017 – 20th International Conference, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 20–37. Springer Verlag, 2017. doi:10.1007/978-3-319-66263-3_2.

9 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining treewidth and backdoors for CSP. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2017.36.

10 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Transactions on Algorithms*, 13(2):29:1–29:32, 2017. Full version of a SODA'16 paper. doi:10.1145/3014587.

11 Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Zivny. Backdoors into heterogeneous classes of SAT and CSP. *J. of Computer and System Sciences*, 85:38–56, 2017. doi:10.1016/j.jcss.2016.10.007.

12 Serge Gaspers, Sebastian Ordyniak, M. S. Ramanujan, Saket Saurabh, and Stefan Szeider. Backdoors to q-Horn. *Algorithmica*, 74(1):540–557, 2016. doi:10.1007/s00453-014-9958-5.

13 Serge Gaspers and Stefan Szeider. Strong backdoors to bounded treewidth SAT. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 489–498. IEEE Computer Society, 2013.

14 Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 89–134. Elsevier, 2008.

15 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

16 Leonid Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

17 Nikolas Mählmann, Sebastian Siebertz, and Alexandre Vigny. Recursive backdoors for SAT. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 73:1–73:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.73.

18 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

19 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combin.*, 27(6):1022–1041, 2006.

20 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

21 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proceedings of SAT 2004 (Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May, 2004, Vancouver, BC, Canada)*, pages 96–103, 2004.

22 Sebastian Ordyniak, Andre Schidler, and Stefan Szeider. Backdoor DNFs. In Zhi-Hua Zhou, editor, *Proceeding of IJCAI-2021, the 30th International Joint Conference on Artificial Intelligence*, pages 1403–1409, 2021. doi:10.24963/ijcai.2021/194.

23 Marko Samer and Stefan Szeider. Backdoor trees. In *AAAI 08, Twenty-Third Conference on Artificial Intelligence, Chicago, Illinois, July 13–17, 2008*, pages 363–368. AAAI Press, 2008.

24 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

25 Thomas J. Schaefer. The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 216–226. ACM, 1978.

26 Ryan Williams, Carla Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003*, pages 1173–1178. Morgan Kaufmann, 2003.