CSP Beyond Tractable Constraint Languages

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

The constraint satisfaction problem (CSP) is among the most studied computational problems. While NP-hard, many tractable subproblems have been identified (Bulatov 2017, Zuk 2017). Backdoors, introduced by Williams, Gomes, and Selman (2003), gradually extend such a tractable class to all CSP instances of bounded distance to the class. Backdoor size provides a natural but rather crude distance measure between a CSP instance and a tractable class. Backdoor depth, introduced by Mählmann, Siebertz, and Vigny (2021) for SAT, is a more refined distance measure, which admits the parallel utilization of different backdoor variables. Bounded backdoor size implies bounded backdoor depth, but there are instances of constant backdoor depth and arbitrarily large backdoor size. Dreier, Ordyniak, and Szeider (2022) provided fixed-parameter algorithms for finding backdoors of small depth into the classes of Horn and Krom formulas.

In this paper, we consider backdoor depth for CSP. We consider backdoors w.r.t. tractable subproblems C_Γ of the CSP defined by a constraint language Γ, i.e., where all the constraints use relations from the language Γ. Building upon Dreier et al.'s game-theoretic approach and their notion of separator obstructions, we show that for any finite, tractable, semi-conservative constraint language Γ, the CSP is fixed-parameter tractable parameterized by the backdoor depth into C_Γ plus the domain size.

With backdoors of low depth, we reach classes of instances that require backdoors of arbitrary large size. Hence, our results strictly generalize several known results for CSP that are based on backdoor size.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases CSP, backdoor depth, constraint language, tractable class, recursive backdoor

Digital Object Identifier 10.4230/LIPIcs.CP.2022.20

Funding *Sebastian Ordyniak*: Supported by the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1).

1 Introduction

To face the NP-completeness of the Constraint Satisfaction Problem (CSP), much effort has been spent in identifying polynomial-time solvable subproblems [5]. Tractability can be reached by

1. restricting the *constraint language* in terms of limiting the relations allowed to be used in constraints (e.g., [3, 6, 10, 30, 33]),
2. restricting the *graphical structure* of how constraints and variables interact (e.g., [7, 21, 22]), or
3. restricting both language and structure with *hybrid restrictions* (e.g., [8, 9, 11]).

Some of the considered restrictions are *gradual* in the sense that they support an infinite chain of classes $\mathcal{C}_0 \subsetneq \mathcal{C}_1 \subsetneq \mathcal{C}_2 \subsetneq \dots$ of instances, where each $\mathcal{C}_i$ can be solved in polynomial time. When the order of polynomial bound on the solving time remains the same for all


© Jan Dreier, Sebastian Ordyniak, and Stefan Szeider;
licensed under Creative Commons License CC-BY 4.0

the $i = 0, 1, 2, \ldots$ one speaks about *fixed-parameter tractability* (FPT) [12, 13, 15, 26, 29]. Most structural restrictions like bounded treewidth or hypertree width are gradual by definition [22, 22]. In contrast, language restrictions tend to be categorical by definition, as either an instance belongs to a class $\mathcal{C}_\Gamma$ defined by a tractable constraint language Γ or it does not.

However, by means of (strong[1]) *backdoors* introduced by Williams, Gomes and Selman [31, 32], one can build a chain $\mathcal{C}_\Gamma = \mathcal{C}_0 \subsetneq \mathcal{C}_1 \subsetneq \mathcal{C}_2 \subsetneq \ldots$ on top of such a class defined by a language. A CSP instance belongs to $\mathcal{C}_i$ if there is a set of i variables, called a backdoor, such that all possible instantiations of these variables move the instance into the base class $\mathcal{C}_\Gamma$. The size of a smallest backdoor provides a *distance measure* between the considered CSP instance and the base class.

The size of a smallest backdoor is a fundamental but still rather crude distance measure. Samer and Szeider [28] therefore proposed *backdoor trees*, where one counts the number of leaves of a decision tree ranging over all the variables of a backdoor; Ordyniak et al. [27] obtained further fixed-parameter tractability results for backdoor trees. A backdoor of size k over a Boolean domain can yield backdoor trees between $k + 1$ and 2^k leaves, and so it is more efficient to minimize the number of leaves than the size. Very recently, in the context of SAT, Mählmann, Siebertz, and Vigny [23] proposed the concept of *backdoor depth*, which extend backdoor trees by adding nodes where the tree branches into connected components. The advantage of considering backdoors of small depth relies on the observation that if an instance decomposes into multiple components, 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 extended decision tree is bounded, one can still utilize it for efficiently solving the instance. In the context of graphs, similar ideas are used in the study of tree-depth [24, 25] and elimination distance [4, 16].

The challenging algorithmic question is to find a backdoor of small depth into a fixed base class, if it exists. Mählmann et al. [23] gave an FPT algorithm for SAT with respect to the base class Null consisting of formulas without variables; any bounded-depth backdoor into that class must contain all the variables of the instance. Already for this simple base class, there are instances of bounded backdoor depth that cannot be efficiently solved by other known methods. Previously [14], we extended this FPT result to bounded-depth backdoors into the classes of Horn (CNF formulas where each clause contains at most one positive literal), dual Horn (each clause contains at most one negative literal), and Krom formulas (each clause contains at most two literals).

Contribution. In this paper, we provide the first positive algorithmic results for utilizing backdoors of bounded depth for CSP. Our main technical result covers all base classes $\mathcal{C}_\Gamma$ described by a finite semi-conservative constraint language Γ. As our main result, we show the following (a formal statement is Corollary 20).

> *For any finite, tractable, semi-conservative constraint language Γ, the CSP is fixed-parameter tractable parameterized by the smallest depth of a backdoor into $\mathcal{C}_\Gamma$ plus the domain size of the instance.*

Thus, we indeed have a chain $\mathcal{C}_\Gamma = \mathcal{C}_0 \subsetneq \mathcal{C}_1 \subsetneq \mathcal{C}_2 \subsetneq \ldots$ on top of any such class $\mathcal{C}_\Gamma$, where $\mathcal{C}_i$ contains instances with a backdoor of depth i, and where the order of the polynomial-time algorithm for solving $\mathcal{C}_i$ is of the same order as the polynomial that bounds the solving time for $\mathcal{C}_\Gamma$.

[1] We focus only on strong backdoors and do not consider weak backdoors.

Backdoor depth can capture and exploit structure in CSP instances that is not captured by any other known method. In the following, we list here some known CSP parameters that admit fixed-parameter tractable CSP solving. For each of these parameters, there are CSP instances for which the parameter can be arbitrarily large, but where Γ-backdoor depth is bounded by a constant:

- backdoor size [20];
- backdoor depth for SAT [14, 23];
- backdoor size into heterogeneous and scattered base classes [19, 20];
- backdoor treewidth [18].

We closely follow the approach we introduced in recent work on SAT [14]. On a high level, we construct backdoors by simultaneously computing an upper bound in the form of an approximate backdoor and a lower bound, using so-called *obstructions*, i.e., parts of the instance that can be proven to be "far away" from the base class. As in our work on SAT, we use two types of obstructions:

1. a slightly modified version of the obstructions trees that have been introduced by Mählmann et al. [23] and
2. a new variant of separator obstructions that we introduced for SAT [14] to allow the handling of base classes that admit arbitrary long paths (in the incidence graph of a CNF formula).

This new variant of separator obstructions is tailor-made for CSP and base classes defined via finite constraint languages. It allows us to improve the algorithm's efficiency, from a triple-exponential run-time dependency to a double-exponential run-time dependence on backdoor depth.

We present our results using the *game-theoretic framework* for backdoor depth that we introduced for SAT [14], which greatly simplifies the presentation of our algorithm.

Due to space constraints, we omit proofs of some technical claims, marked ($\star$).

2 Preliminaries

2.1 CSP

Let D be a set and n and n' be non-negative integers. An n-ary relation on D is a subset of D^n. For a tuple $t \in D^n$, we denote by $t[i]$, the i-th entry of t, where $1 \leq i \leq n$. For two tuples $t \in D^n$ and $t' \in D^{n'}$, we denote by $t \circ t'$, the concatenation of t and t'.

An instance of a *constraint satisfaction problem* (CSP) I is a triple $\langle V, D, C\rangle$, where V is a finite set of variables over a finite set (domain) D, and C is a set of constraints.We assume that D is given explicitly as a list of all domain values. A *constraint* $c \in C$ consists of a *scope*, denoted by $V(c)$, which is an ordered list of a subset of V, and a relation, denoted by $R(c)$, which is a $|V(c)|$-ary relation on D; $|V(c)|$ is the *arity* of c. To simplify notation, we sometimes treat ordered lists without repetitions, such as the scope of a constraint, like sets. For a variable $v \in V(c)$ and a tuple $t \in R(c)$, we denote by $t[v]$, the i-th entry of t, where i is the position of v in $V(c)$. For a CSP instance $I = \langle V, D, C\rangle$ we sometimes denote by $V(I)$, $D(I)$, and $C(I)$, its set of variables V, its domain D, and its set of constraints C, respectively. We usually assume, w.l.o.g, that each variable in $V(I)$ appears in the scope of at least one constraint in $C(I)$. The *size* $|I|$ of a CSP instance I is the sum of the sizes of its constraints, where the size of a constraint of arity a with t tuples and domain size δ is $at \log \delta$. A *solution* to a CSP instance I is a mapping $\tau : V \to D$ such that $\langle \tau(v_1), \ldots, \tau(v_{|V(c)|})\rangle \in R(c)$ for every $c \in C$ with $V(c) = \langle v_1, \ldots, v_{|V(c)|}\rangle$. A CSP instance is *satisfiable* if and only if it has at least one solution.

Let $V' \subseteq V$ and $\tau : V' \to D$. For a constraint $c \in C$, we denote by $c[\tau]$, the constraint whose scope is $V(c) \setminus V'$ and whose relation contains all $|V(c[\tau])|$-ary tuples t such that there is a $|V(c)|$-ary tuple $t' \in R(c)$ with $t[v] = t'[v]$ for every $v \in V(c[\tau])$ and $t'[v] = \tau(v)$ for every $v \in V' \cap V(c)$. We denote the assignment $\tau : \{x\} \to D$ with $\tau(x) = q$ simply by $x = q$.

A constraint $c \in C(I)$ of arity a is *tautological* if it contains all the $|D|^a$ possible tuples. Obviously, removing a tautological constraint from a CSP instance does not change its satisfiability. We denote by $I[\tau]$ the CSP instance with variables $V \setminus V'$, domain D, and constraints $C[\tau]$, where $C[\tau]$ contains all *non-tautological* constraints $c[\tau]$ for every $c \in C$. We would like to point out that the removal of tautological constraints is important in the context of backdoor depth as it makes the notion more powerful.

Let $\tau_1 : V_1 \to D$ and $\tau_2 : V_2 \to D$ be two assignments. We say that the two assignments are *compatible* if $\tau_1(v) = \tau_2(v)$ for every $v \in V_1 \cap V_2$. Moreover, if τ_1 and τ_2 are compatible, we denote by $\tau_1 \cup \tau_2$ the assignment $\tau : V_1 \cup V_2 \to D$ given by $\tau(v) = \tau_1(v)$ if $v \in V_1$ and $\tau(v) = \tau_2(v)$ if $v \in V_2$.

The *incidence graph* of a CSP instance I is the bipartite graph G_I whose vertices are the variables and constraints of I, and where a variable x and a constraint c are adjacent if and only if $x \in V(c)$. Via incidence graphs, graph theoretic concepts directly translate to CSP instances. For instance, we say that I is connected if G_I is connected, and I' is a *connected component* of I if $D(I') = D(I)$, and where $V(I')$ and $C(I')$ are maximal subsets of $V(I)$ and $C(I)$, respectively, such that G_I is connected. $\mathrm{Conn}(I)$ denotes the set of connected components of I. Occasionally, we will also consider the *primal graph* of a CSP instance I, which has as vertex set $V(I)$, and has pairs of variables adjacent if they appear together in the scope of a constraint.

A *constraint language* Γ over a domain D is a set of relations over D. $D(\Gamma)$ is the set of all the elements appearing in the relations in Γ. We denote by $\mathrm{arity}(\Gamma)$ the maximum arity of any relation in Γ. $\mathcal{C}_\Gamma$ denotes the class of CSP instances I with the property that for each $c \in C(I)$ we have $R(c) \in \Gamma$. $\mathrm{CSP}(\Gamma)$ refers to the CSP with instances restricted to $\mathcal{C}_\Gamma$. A constraint language Γ is *tractable* if $\mathrm{CSP}(\Gamma)$ can be solved in polynomial time, Γ is *linear-time tractable* if $\mathrm{CSP}(\Gamma)$ can be solved in linear time.

Γ is *semi-conservative* (or *1-conservative*) [1, 17], if for each $q \in D(\Gamma)$ one can express with Γ the unary constraint $x = q$; more precisely, there is a satisfiable instance I_q of $\mathrm{CSP}(\Gamma)$ and a variable $x \in V(I_q)$ such that for each solution τ of I_q we have $\tau(x) = q$. Semi-conservative constraint languages are very natural, as one would expect in any reasonable practical settings that the unary relations are present. Indeed, some authors (e.g., [10]) even define the CSP so that every variable can have its own set of domain values, making (semi-)conservativeness a built-in property.

A constraint language Γ is *closed under assignments* if for every constraint c with $R(c) \in \Gamma$ and every assignment τ, it holds that $R(c[\tau]) \in \Gamma$. For a constraint language Γ we denote by Γ^* the smallest constraint language that contains Γ and is closed under assignments.

▶ **Lemma 1** ([17]). *If a semi-conservative constraints language Γ is tractable, then Γ^* is also tractable.*

We would like to point out that the original definition of a backdoor by Williams et al. [31, 32] assumes the base class to be closed under assignments. Hence, it is natural to assume this property in the context of backdoor depth, directly or indirectly by means of semi-conservativeness of the considered language.

2.2 Backdoors

Backdoors are defined relative to some fixed *base class* $\mathcal{C}$ of instances of the problem under consideration (i.e., CSP), for which satisfiability and membership in $\mathcal{C}$ are polynomial-time decidable. In the context of CSP, we define a *$\mathcal{C}$-backdoor set* of a CSP instance I as a set $B \subseteq V(I)$ of variables such that $I[\tau] \in \mathcal{C}$ for every $\tau : B \to D(I)$. For a constraints language Γ, we usually denote the base class $\mathcal{C}_\Gamma$ by Γ itself. Thus, for example, instead of $\mathcal{C}_\Gamma$-backdoors, we talk of Γ-backdoors. If we know a $\mathcal{C}$-backdoor set B of I, we can reduce the satisfiability of I to the satisfiability of $|D(I)|^{|B|}$ CSP instances in $\mathcal{C}$. The challenging problem is to find a $\mathcal{C}$-backdoor set of a given instance that reduces the satisfiability problem to instances from $\mathcal{C}$.

3 Backdoor Depth

Component backdoor trees generalize backdoor trees as considered by Samer and Szeider [28] by allowing an additional type of nodes, *component nodes*, where the current instance is split into connected components. More precisely, let $\mathcal{C}$ be a class of CSP instances (called the base class) and I a CSP instance. A *component $\mathcal{C}$-backdoor tree for I* is a pair (T, φ), where T is a rooted tree and φ is a mapping that assigns each node t a CSP instance $\varphi(t)$ such that the following conditions are satisfied:

1. For the root r of T, we have $\varphi(r) = I$.
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. $D(I) = \{q_1, \ldots, q_\delta\}$ and t has exactly δ children $t_1, \ldots, t_\delta$ where for some variable $x \in V(\varphi(t))$ we have $\varphi(t_i) = \varphi(t)[x = q_i]$; in this case we call t a *variable node*.
   b. $\mathrm{Conn}(\varphi(t)) = \{I_1, \ldots, I_k\}$ for $k \geq 2$ and t has exactly k children $t_1, \ldots, t_k$ with $\varphi(t_i) = I_i$; in this case we call t a *component node*.

Thus, a backdoor tree as considered by Samer and Szeider [28] is just a component backdoor tree without component nodes. The *depth* of a backdoor is the largest number of variable nodes on any root-leaf path in the tree. The *$\mathcal{C}$-backdoor depth* $\mathrm{depth}_\mathcal{C}(I)$ of an instance I into a base class $\mathcal{C}$ is the smallest depth over all component $\mathcal{C}$-backdoor trees of I. If $\mathcal{C}$ is defined in terms of a constraint language Γ, we simply write $\mathrm{depth}_\Gamma(I)$.

Alternatively, we can define $\mathcal{C}$-backdoor depth recursively as follows:

$$\mathrm{depth}_\mathcal{C}(I) := \begin{cases} 0 & \text{if } I \in \mathcal{C}; \\ 1 + \min_{x \in V(I)} \max_{a \in D(I)} \mathrm{depth}_\mathcal{C}(I[x = a]) & \text{if } I \notin \mathcal{C} \text{ and } I \text{ is connected}; \\ \max_{I' \in \mathrm{Conn}(I)} \mathrm{depth}_\mathcal{C}(I') & \text{if } I \notin \mathcal{C} \text{ and } I \text{ is not connected.} \end{cases}$$

▶ **Lemma 2.** *ategory=ssrbd,normal]lemma Let Γ be a constraint language such that $\mathcal{C}_\Gamma$ can be solved in time $\mathcal{O}(n^c)$ for some constant $c \geq 1$ and input size n. Assume we are given a CSP instance I whose size is m, $\delta = |D(I)|$, and a component Γ-backdoor tree (T, φ) of I of depth d. Then, we can solve I in time $\mathcal{O}((\delta^d m)^c)$.*

Proof. We start by showing that $\sum_{\ell \in L(T)} |\varphi(\ell)| \leq \delta^d m$, where $L(T)$ denotes the set of leaves of T, using induction on d and m. The statement holds if $d = 0$ or $m \leq 1$. We show that it also holds for larger d and m. If the root is a variable node, then it has δ children $c_1, \ldots, c_\delta$, and the subtree rooted at any of these children represents a component backdoor tree for the CSP instance $\varphi(c_i)$ of depth $d - 1$. Therefore, by the induction hypothesis, we obtain that $s_i = \sum_{\ell \in L(T_i)} |\varphi(\ell)| \leq \delta^{d-1} m$, for every subtree T_i rooted at c_i. Consequently, $\sum_{\ell \in L(T)} |\varphi(\ell)| = \sum_{1 \leq i \leq \delta} s_i \leq \delta \delta^{d-1} m = \delta^d m$, as required. If, on the other hand, the root is

a component node, then its children, say $c_1, \ldots, c_k$, are labeled with CSP instances of sizes $m_1 + \cdots + m_k = m$. Therefore, for every subtree T_i of T rooted at c_i, we have that T_i is a component backdoor tree of depth d for $\varphi(c_i)$, which using the induction hypothesis implies that $\sum_{\ell \in L(T_i)} |\varphi(\ell)| \leq \delta^d m_i$. Hence, we obtain $\sum_{\ell \in L(T)} |\varphi(\ell)| \leq \delta^d m$ in total.

To solve the CSP instance I, we first solve all CSP instances associated with the leaves of T. Because, as shown above, their total size is at most $\delta^d m$, this can be achieved in time $\mathcal{O}((\delta^d m)^c)$, because $\mathcal{C}_\Gamma$ can be solved in time $\mathcal{O}(n^c)$ for some constant $c \geq 1$ and input size n. Let us call a leaf true/false if and only it is labeled by a satisfiable/unsatisfiable CSP instance, respectively. We now propagate the truth values upwards to the root, considering a component node as the logical *and* of its children, and the a variable node as the logical *or* of its children. I is satisfiable if and only if the root of T is true. We can carry out the propagation in time linear in the number of nodes of T, which is linear in the number of leaves of T, i.e., at most $\delta^d m$. ◀

4 Technical Overview

On a high level, the approach of our algorithm is similar to the approach we employed for SAT [14]. The critical difference lies in the exact definition of separator obstructions in Section 5, which we adapt to CSP and base classes defined via finite constraint languages. Apart from lifting the approach from SAT to CSP, our tailor-made separator obstructions also allow us to obtain a more efficient algorithm. As our first order of business, we state an equivalent formulation of backdoor depth using *connector-splitter games*, as we previously introduced for SAT [14], allowing us to greatly simplify the exposition of our algorithm.

▶ **Definition 3.** *Let Γ be a finite constraint language that is closed under assignments and let $I = \langle V, D, C \rangle$ be a CSP instance. We denote by* Game(I, Γ) *the so-called* Γ-backdoor depth game on I. *The game is played between two players, the* connector *and the* splitter. *The* positions *of the game are CSP instances. At first, the connector chooses a connected component of I 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.*

- *The splitter chooses a variable $v \in V(J)$.*
- *The connector chooses an assignment $\tau \colon \{v\} \to D$ 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 $\mathcal{C}_\Gamma$, 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 definitions of the game and the fact that the (strategy) tree obtained by playing all possible plays of the connector against a given strategy for the splitter forms a component backdoor tree 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 4.** *The splitter has a strategy for the game* Game(I, Γ) *to reach within at most d rounds a winning position if and only if I has a Γ-backdoor of depth at most d.*

Backdoor depth games mean that we no longer have to explicitly construct a backdoor. Instead, in Section 6, we compute winning strategies for the splitter, which appear to be easier to reason about. Such a strategy can then be automatically converted into a backdoor algorithm (Lemma 6).

We start by describing these so called *splitter-algorithms* and how they can be turned into an algorithm to compute backdoor depth. The algorithms will have some auxiliary internal state that they modify with each move. Formally, a *splitter-algorithm* for a game $\textsc{Game}(I, \Gamma)$, where Γ is a finite constraint language that is closed under assignments, 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.

Suppose we have a game $\textsc{Game}(I, \Gamma)$ 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, an internal state is initialized with S (if no additional input is given, it 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 5.** *We say a splitter-algorithm* implements a strategy to reach for a game $\textsc{Game}(I, \Gamma)$ 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 $\textsc{Game}(I, \Gamma)$ 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.*

Using the following observation converts splitter-algorithms into algorithms for backdoors. It builds backdoors by always trying out all the next moves of the connector.

▶ **Lemma 6 (⋆).** *Assume there exists a function $f(d, \Gamma)$ and a splitter-algorithm that implements a strategy to reach for each game $\textsc{Game}(I, \Gamma)$ and non-negative integer d within at most $f(d, \Gamma)$ rounds either a winning position or (an internal state representing) a proof that the Γ-backdoor depth of I is larger than d.*

Further assume this splitter-algorithm always takes at most $\mathcal{O}(|I|)$ time to compute its next move. Then there exists an algorithm that, for a CSP instance I, a finite constraint language Γ that is closed under assignments, and a non-negative integer d in time at most $|D(I)|^{2f(d,\Gamma)}\mathcal{O}(|I|)$ either returns a component Γ-backdoor tree of depth at most $f(d, \Gamma)$ or concludes that the Γ-backdoor depth of I is larger than d.

For improved 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 19) that either concludes that there is no backdoor of depth d, or computes a component backdoor tree of depth at most $2^{\mathcal{O}(d)}$. Using Lemma 6, we see that this is equivalent to a splitter-algorithm that plays for $2^{\mathcal{O}(d)}$ rounds to either reach a winning position or a proof that the backdoor depth is larger than d.

Here and in the following, we say that a constraint is Γ-bad for a finite constraint language Γ if its relation is not in Γ; otherwise we say that the constraint is Γ-good. Note that if Γ is close under assignments, then a Γ-good constraint remains Γ-good even after assigning additional variables and a conversely a constraint that is Γ-bad in some subinstance obtained by assigning some variables is also Γ-bad in the original instance.

Our proofs of high backdoor depth come in the form of so-called *obstruction trees*, which have first been introduced by Mählmann et al. [23]. These are trees in the incidence graph of a CSP instance. Their node set therefore consists of both variables and constraints. 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 Γ-bad constraint 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. Because 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 7.** *Let I be a CSP instance and Γ be a constraint language that is closed under assignments. We inductively define Γ-obstruction trees T of increasing depth.*

- *Let c be a Γ-bad constraint of I. The set $T = \{c\}$ is a* Γ-obstruction tree in I of depth 0.
- *Let T_1 be a Γ-obstruction tree of depth i in I. Let β be a partial assignment of the variables in I. Let T_2 be an obstruction tree of depth i in $I[\beta]$ such that that no variable $v \in V(I[\beta])$ is contained both in a constraint of T_1 and T_2. Let further P be a path (in the incidence graph) connecting T_1 and T_2 in I. Then $T = T_1 \cup T_2 \cup V(P) \cup C(P)$ is a* Γ-obstruction tree in I of depth $i+1$.

▶ **Lemma 8** ($\star$)**.** *Let I be a CSP instance and Γ be a constraint language that is closed under assignments. If there is a Γ-obstruction tree of depth d in I, then the Γ-backdoor depth of I is at least $d+1$.*

Our splitter-algorithm will construct obstruction trees of increasing depth by a recursive procedure (Lemma 18) that we outline now. We say a splitter-algorithm satisfies *property i* if it reaches in each game $\textsc{Game}(I, \Gamma)$ within $g_{\mathcal{C}}(i, d)$ rounds (for some function $g_{\mathcal{C}}(i, d)$) either

- **i)** a winning position, or
- **ii)** a position J and a Γ-obstruction tree T of depth i in I such that no variable in $var(J)$ occurs in a constraint of T, or
- **iii)** a proof that the Γ-backdoor depth of I is at least d.

A splitter-algorithm satisfying property $d+1$ then directly implies our main result, the approximation algorithm for backdoor depth, using Lemma 8 and Lemma 6. 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 Γ-backdoor depth of I is at least d, we are done. Let us assume this does not happen, so we can focus on the much more interesting case 2).

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 T_1 and T_2 are in the same component of F, 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 $I[\beta]$. In order to join both trees together into an obstruction of depth i, we have to show, according to Definition 7 that no variable $v \in var(I[\beta])$ occurs both in a constraint of T_1 and T_2. Since no variable in $var(J_1)$ occurs in a constraint 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 (details also in proof of Lemma 18).

Finally, we need to ensure is that we reach a position J such that no variable in $var(J)$ occurs in a constraint 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.

It is important to note here, that the exact notion of "separation" between obstruction trees plays a crucial role and is one of the main differences between the approaches used by Mählmann et al. [23] and Dreier et al. [14]. The former 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 formulas (or equivalently backdoor depth to some finite constraint language of arity at most two), this approach cannot work since obstruction trees for Krom formulas can have arbitrarily many clauses. We solve this issue by adapting the separator obstructions in [14] from SAT to CSP. We also exploit the fact that our base classes have bounded arity (in contrast to, e.g., the class of Horn formulas) to simplify their separator obstructions significantly allowing us to drop the complexity for solving CSP using backdoor depth from triple to double exponential in the backdoor depth.

5 Separator Obstructions

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 therefore the splitter cannot simply select all variables in (constraints of) P. Instead, given such a path P that we want to separate, we will use separator obstructions to develop a splitter-algorithm (Lemma 16) that reaches in each game $\textsc{Game}(I, \Gamma)$ within a bounded number of rounds either

- **i)** a winning position, or
- **ii)** a position J such that no variable in $var(J)$ occurs in a constraint of P, or
- **iii)** a proof that the backdoor depth of I 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) Γ-bad constraint 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 the tree T_ℓ has maximum degree at most three and that every *outside* variable, i.e., any variable that is not an important variable assigned by τ, can occur in at most four constraints of T_ℓ. Therefore assigning any outside variable can split T_ℓ in only constantly many parts. Together with the assignment τ, which we will use as a guide for the connector for the variables inside the obstruction, this will allow us to show that the connector can play in such a way that after every round at least a constant fraction of the separator obstruction remains 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 $\textsc{Game}(I, \Gamma)$ and already has built a separator obstruction $X = \langle \langle P_1, \dots, P_i \rangle, \tau \rangle$ containing Γ-bad constraints $b_1, \dots, b_i$; note

that τ is compatible with τ_J. If J is already a winning position, then we are done. Therefore, J has to contain a Γ-bad constraint. If no Γ-bad constraint has a path to T_i in J, then J satisfies 2) and we are also done. Otherwise, let b_{i+1} be a Γ-bad constraint 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 in such a way 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 Γ-bad constraints $b_1, \ldots, b_{i+1}$ Γ-good. Therefore, a natural choice is all variables of b_{i+1} to X, i.e., to make those variables important. The following lemma shows that this is possible, because the number of those variables is bounded.

▶ **Lemma 9** ($\star$). *Let I be a CSP instance and Γ be a finite constraint language. If I has Γ-backdoor depth at most some integer d, then every constraint of I has arity at most $d + \mathrm{arity}(\Gamma)$.*

The next thing that we need to ensure is that any outside variable can not destroy too many paths. Note that by choosing a *shortest* path P_{i+1}, we have already ensured that no variable occurs on more than two constraints 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, we know that every variable that occurs on T_i and on P_{i+1} must occur in the constraint c in P_{i+1} that is closest to T_i but not in T_i itself. Similarly, to how we dealt with the Γ-bad constraints, we will now add all variables that occur in c to X. This ensures that no outside variable can occur in both T_i and P_{i+1} , which (by induction over i) implies that every outside variable occurs in at most two constraints (either from T_i or from P_{i+1}). However, since removing any single constraint can still be arbitrarily bad if the constraint has a high degree in the separator obstruction, we further need to ensure that all constraints of the separator obstruction have small degree. We achieve this by adding the variables occurring in any constraint 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 constraint. 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 10.** *A* Γ-separator obstruction *for I is a tuple $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ satisfying the following conditions.*

- *P_1 is a shortest Γ-good path between two Γ-bad constraints b_0 and b_1 in I.*
- *For $1 \leq i \leq \ell$, let $T_i = \bigcup_{1 \leq j \leq i} P_j$.*
- *For every $i \in [2, \ell]$, P_i is a shortest Γ-good path from T_{i-1} to a Γ-bad constraint b_i that is closest to T_{i-1} in $I[\tau_{i-1}]$, where for every $i \in [0, \ell]$, τ_i is the restriction of τ to the variables in V_i given below.*
- *For every $i \in [2, \ell]$, let e_i be the constraint in $P_i \setminus T_{i-1}$ that is closest to T_{i-1} and let f_i be the constraint $P_i \cap T_{i-1}$ if $P_i \cap T_{i-1}$ is a constraint, otherwise we set $f_i = e_i$. Moreover, for every $i \in [\ell]$, let B_i be the set $\{b_0, b_1, b_2, e_2, f_2, \ldots, b_i, e_i, f_i\}$ of constraints and let V_i be the set of all variables occurring in any constraint in B_i. Then, τ_i is the restriction of τ to V_i and τ assigns exactly the variables in V_ℓ.*

We define the size *of X to be the number of leaves of $T = T_\ell$.*

We start by showing some simple but important properties of separator obstructions.

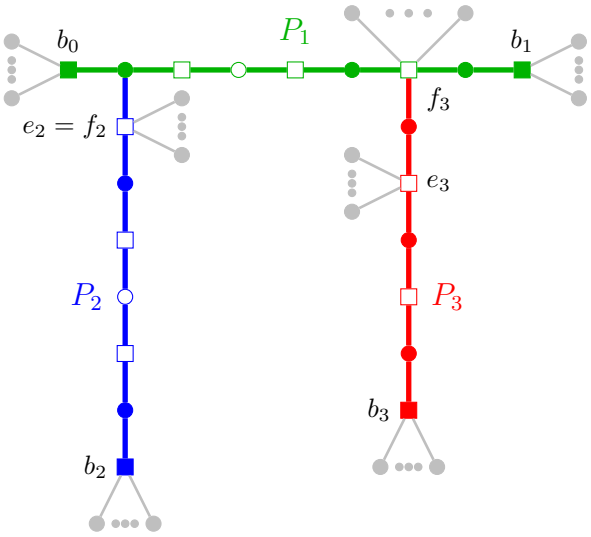

Figure 1 A separator obstruction containing three paths P_1, P_2, and P_3. The figure shows the vertices and edges of the incident graph. Variables are represented by circles and constraints are represented by rectangles. Filled variables are contained in V_3 (all other variables are not) and filled rectangles are bad constraints (all other constraints are good). Only the black variables and edges are part of the tree of the separator obstruction, grey variables and edges are not part of the tree but are part of V_3.

▶ **Lemma 11** (⋆). *Let Γ be a finite constraint language that is closed under assignments and let $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ be a Γ-separator obstruction in I, then for every $i \in [\ell]$:*

(P1) *T_i is a tree with leaves $b_0, \ldots, b_i$.*

(P2) *T_i has maximum degree at most 3.*

(P3) *Every variable $v \notin V_i$ is contained in at most two constraints of T_i and moreover those constraints are consecutive in T_i.*

(P4) *Every variable $v \in V_i \setminus V_{i-1}$ is contained in most 4 constraints of T_i.*

(P5) *If β is any assignment compatible with τ that does not assign any variable in $V_i \setminus V_{i-1}$, then b_i is a Γ-bad constraint in $I[\beta]$.*

Our next aim is to show that separator obstructions – just like obstruction trees – can be employed to obtain a lower bound on the backdoor depth of a CSP instance. For this it is important to show that assigning a single variable cannot sufficiently destroy a separator obstruction.

Note that Lemma 11 already provides a first step in this direction. In particular, (P3) limits the influence of variables outside of V_ℓ to only two constraints and (P4) limits the influence of variables inside V_ℓ, at least towards the part of the separator obstruction that was constructed before the variable was added. To limit the influence of variables in V_ℓ also on the remaining part of the separator obstruction, we show that even though these variables can appear in arbitrary many constraints of the remaining part, their influence is still limited as long as we only consider CSP instances obtained by assigning those variables according to τ.

▶ **Definition 12.** *Let $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ be a Γ-separator obstruction for I and let β be an assignment that is compatible with τ. Moreover, let c be a constraint contained in T and let i be minimal such that c is contained in T_i. We say that c is* tainted *by β, if $V(\beta)$ contains a*

variable v in the scope of c such that $v \notin V_{i-1}$. Otherwise we say that c is untainted *by β. Similarly, we say that a subtree T' is untainted by β if so is every constraint of T' and moreover $V(\beta)$ does not contain a variable of T'.*

▶ **Lemma 13** (⋆). *Let Γ be a finite constraint language that is closed under assignments and let $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ be a Γ-separator obstruction in I, let β be an assignment that is compatible with τ, and let T' be a subtree of T untainted by β. Then, $I[\beta]$ contains T'.*

▶ **Lemma 14** (⋆). *Let T be a tree with n leaves and maximum degree g and let $R \subseteq V(T)$. Then, $T - R$ has a component containing at least $(n - |R|)/(g|R|)$ leaves of T.*

We are now ready to show our main result of this subsection, namely, that separator obstructions can be used to obtain lower bounds on the backdoor depth of a CSP instance.

▶ **Lemma 15.** *Let Γ be a finite constraint language that is closed under assignments and let I be a CSP instance. If I has a Γ-separator obstruction of size at least $n = (d+2)(15)^d$, then I has Γ-backdoor depth at least d.*

Proof. Let $X = \langle\langle P_1, \ldots, P_\ell\rangle, \tau\rangle$ be a Γ-separator obstruction for I of size at least $n = (d+2)(15)^d$ and let $B = \{b_0, \ldots, b_\ell\}$.

Consider the following strategy S for the connector in the game $\textsc{Game}(I, \Gamma)$. Suppose that we have reached position J in the game and suppose that the splitter chooses a variable v as his next move. We distinguish the following two cases:

1. If $v \notin V_\ell$, then the connector plays an arbitrary assignment α for v and chooses a component of $J[\alpha]$ containing a subtree untainted by $\tau_J \cup \alpha$ of T containing the largest subset of B among all components of $J[\alpha]$.
2. If $v \in V_\ell$, then the connector plays the assignment $\alpha(v) = \tau(v)$ for v and chooses the component of $J[\alpha]$ containing a subtree of T untainted by $\tau_J \cup \alpha$ containing the largest subset of B among all components of $J[\alpha]$.

Let J be a position reached in the game $\textsc{Game}(I, \Gamma)$ against S at round i. We show by induction on i that J contains a subtree of T untainted by τ_J containing at least $n_i = n/(15^i) - 1$ elements from B; note that because of Lemma 11 (P1) the elements of B contained in the subtree are the leaves of the subtree.

The claim clearly holds for $i = 0$ since the connector chooses the component of I containing T. Moreover, for $i > 0$ let J' be the predecessor (position) of J in $\textsc{Game}(I, \Gamma)$. By the induction hypothesis J' contains a subtree T' of T untainted by $\tau_{J'}$ containing at least $n_{i-1} = n/(15^{i-1}) - 1$ elements from B. Let v be the variable chosen by the splitter at position J' and let α be the assignment of v chosen by the connector.

If $v \notin V_\ell$, then it follows from Lemma 11 (P3) with $i = \ell$ that v is contained in at most 2 constraints of T and therefore α can taint at most 2 constraints of T. Otherwise let $1 \leq i \leq \ell$ be minimal such that $v \in V_i$. Assume for contradiction α taints a constraint c in $T \setminus T_i$. Then let j be minimal such that c is contained in T_j. Obviously, $j > i$. But then $v \notin V_{j-1}$, a contradiction to our choice of i. This means α cannot taint any constraints in $T \setminus T_i$. Since $1 \leq i \leq \ell$ is minimal with $v \in V_i$, we have $v \in V_i \setminus V_{i-1}$ and by (P4) v is contained in at most 4 constraints of T_i. This means α can taint at most 4 constraints of T_i. In total, α can taint at most 4 constraints of T and therefore also of T'. Further, since T' is untainted by τ_J and $\tau_{J'} = \tau_{J'} \cup \alpha$, the assignment τ_J taints at most 4 constraints of T'.

Moreover, because of Lemma 13 and the fact that $\tau_{J'} \cup \alpha$ is compatible with τ, it follows that every subtree of T' untainted by α is contained in some connected component of $J'[\alpha]$. Since T' has maximum degree at most 3 due to Lemma 11 (P2), we obtain from

Observation 14 that after removing the at most 4 constraints together with the variable v from J', there is a component of $J'[\alpha]$ containing a subtree of T' untainted by τ_J with at least $(n_{i-1}-5)/(3\cdot 5) = (n_{i-1}/15) - 1/3 \geq (n/15^{i-1}-1)/15 - 1/3 = n/15^i - 2/5 \geq n_i$ elements of B. Since the connector will choose such a component this concludes the proof of the claim.

Therefore, we obtain that if J is a position reached after i rounds in the game $\textsc{Game}(I,\Gamma)$ against S, then J contains a subtree of T untainted by τ_J containing at least $n_i = n/(15)^i - 1$ constraints from B. In particular, this implies that if J is a position reached after d rounds against S, then J contains a subtree of T untainted by τ_J containing at least $n/(15)^d - 1 = d+1$ constraints from B. Finally, because of Lemma 11 (P5) at least one of these constraints is Γ-bad in J, which concludes the proof of the lemma. ◀

6 Winning Strategies and Algorithms

In this section, we will present our algorithmic results. In Section 4, 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 constraint of P, we are guaranteed that all future obstructions are separated from P, as future obstructions will only contain constraints and variables from J.

▶ **Lemma 16.** *Let Γ be a finite constraint language that is closed under assignments. There exists a splitter-algorithm that implements a strategy to reach for each game $\textsc{Game}(I,\Gamma)$, non-negative integer d, and shortest Γ-good path P between two Γ-bad constraints in I within at most $(3\cdot \mathrm{arity}(\Gamma)+d)(d+2)(15)^d$ rounds either:*

- **i)** *a winning position, or*
- **ii)** *a position J such that no variable in $V(J)$ is contained in a constraint of P, or*
- **iii)** *a proof that the Γ-backdoor depth of I is larger than d.*

This algorithm takes at most $\mathcal{O}(|I|)$ time per move.

Proof. Let $X = \langle\langle P_1,\dots,P_\ell\rangle,\tau\rangle$ be a Γ-separator obstruction for I and let τ' be a sub-assignment of τ assigning at least all variables in $V_{\ell-1}$. Then, we call $X = \langle\langle P_1,\dots,P_\ell\rangle,\tau'\rangle$ a *partial Γ-separator obstruction* for I.

Consider the following splitter-algorithm, where for each position J of the game $\textsc{Game}(I,\Gamma)$, we additionally associate a partial Γ-separator obstruction denoted by $X(J) = \langle\langle P_1,\dots,P_\ell\rangle,\tau_J\rangle$ with $P_1 = P$ to every position J. We set $X(S) = \langle\langle P\rangle,\emptyset\rangle$ for the starting position S of the game.

Then, the splitter-algorithm does the following for a position J in $\textsc{Game}(I,\Gamma)$. If $X(J) = \langle\langle P_1,\dots,P_\ell\rangle,\tau_J\rangle$ and there is at least one variable in $V_\ell \setminus V_{\ell-1}$ (assuming that $V_0 = \emptyset$) that has not yet been assigned by τ_J, then the splitter chooses any such variable. Otherwise, $X(J)$ is a Γ-separator obstruction and we distinguish the following cases:

1. If there is a Γ-bad constraint in J that has a path to some vertex of T_ℓ, then let b be a Γ-bad constraint that is closest to any vertex of T_ℓ in J and let P' be a shortest Γ-good path from b to some vertex of T_ℓ in J. Note that $X = \langle\langle P_1,\dots,P_\ell,P_{\ell+1}\rangle,\tau_J\rangle$, where $P_{\ell+1} = P'$, is a partial Γ-separator obstruction for I. The splitter now chooses any variable in $V_{\ell+1}\setminus V_\ell$ and assigns $X' = \langle\langle P_1,\dots,P_\ell,P_{\ell+1}\rangle,\tau_{J'}\rangle$ to the position J' resulting from this move.

2. Otherwise, $X(J)$ can no longer be extended and either: (a) there is no Γ-bad constraint in J, in which case we reached a winning position, i.e., we achieved case i), or (b) every Γ-bad constraint of J has no path to T_ℓ, which implies that no variable of J is contained in a constraint of T_ℓ and therefore also of P, i.e., we achieved case ii).

This completes the description of the splitter-algorithm. Moreover, if every play against the splitter-algorithm ends after at most $(3 \cdot \text{arity}(\Gamma) + d)(d+2)(15)^d$ rounds, every position is either of type i) or type ii) and we are done.

Otherwise, there is a position J that is reached after playing at least $(3 \cdot \text{arity}(\Gamma) + d)(d+2)(15)^d$ rounds. Then, $X(J)$ has size at least $(d+2)(15)^d$ because the size of the Γ-separator obstruction increases by at least 1 after at most $3 \cdot \text{arity}(\Gamma) + d$ steps. This is because every time the Γ-separator obstruction increases by 1, we only add the at most $\text{arity}(\Gamma) + d + 1$ variables of at most one Γ-bad constraint b_i (because of Lemma 9) and the at most $2 \cdot \text{arity}(\Gamma)$ variables of at most two Γ-good constraints (e_i and f_i). Therefore, it follows from Lemma 15 that I has Γ-backdoor depth at least d.

Finally, the splitter-algorithm takes time at most $\mathcal{O}(|I|)$ per round since a Γ-bad constraint that is closest to the current Γ-separator obstruction and the associated shortest path can be found using a simple breadth-first search. ◀

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

▶ **Corollary 17.** *Consider a finite constraint language Γ that is closed under assignments, a game $\textsc{Game}(I, \Gamma)$ and a position J' in this game, a non-negative integer d and shortest Γ-good path P between two Γ-bad constraints in I. There exists a splitter-algorithm that implements a strategy that continues the game from position J' and reaches within at most $(3 \cdot \text{arity}(\Gamma) + d)(d+2)(15)^d$ rounds either:*

- **i)** *a winning position, or*
- **ii)** *a position J such that no variable in $V(J)$ is contained in a constraint of P, or*
- **iii)** *a proof that the Γ-backdoor depth of I is larger than d.*

This algorithm takes at most $\mathcal{O}(|I|)$ 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 18** ($\star$)**.** *Let Γ be a finite constraint language that is closed under assignments. There is a splitter-algorithm that implements a strategy to reach for a game $\textsc{Game}(I, \Gamma)$ and non-negative integers i and d with $1 \leq i \leq d$ within at most $(2^{i+1} - 1)(3 \cdot \text{arity}(\Gamma) + d)(d+2)(15)^d$ rounds either:*

- **i)** *a winning position, or*
- **ii)** *a position J and a Γ-obstruction tree T of depth i in I such that no variable in $V(J)$ is contained in a constraint of T, or*
- **iii)** *a proof that the Γ-backdoor depth of I is larger than d.*

This algorithm takes at most $\mathcal{O}(|I|)$ time per move.

Given Lemma 18, the remaining results now follow easily.

▶ **Theorem 19.** *Let Γ be a finite constraint language that is closed under assignments. We can, for a given CSP instance I and a non-negative integer d, in time at most $|D(I)|^{2^{\mathcal{O}(d)}}|I|$ either:*

1. *compute a component Γ-backdoor tree of I of depth at most $2^{\mathcal{O}(d)}$, or*
2. *conclude that the Γ-backdoor depth of I is larger than d.*

Proof. An obstruction tree of depth d is a proof that the backdoor depth is higher than d, thus for the case $i = d$ the output of the splitter-algorithm in Lemma 18 after $2^{\mathcal{O}(d)}$ rounds reduces to either a winning position, or a proof that the Γ-backdoor depth of I is larger than d. The algorithm takes at most $\mathcal{O}(|I|)$ time per move. The statement then follows from Lemma 6. ◀

▶ **Corollary 20.** *Let Γ be a tractable constraint language that is finite and semi-conservative. The CSP can be solved in time $\delta^{2^{\mathcal{O}(d)}}(|I|)^{\mathcal{O}(1)}$ for instances I with $\delta = |D(I)|$ and $d = \mathrm{depth}_\Gamma(I)$.*

Proof. According to Lemma 1, the closure Γ^* of Γ is also tractable. Furthermore, Γ^* is more permissive than Γ and therefore $\mathrm{depth}_{\Gamma^*}(I) \leq \mathrm{depth}_\Gamma(I) = d$. We use Theorem 19 to compute a component Γ^*-backdoor tree of depth $2^{\mathcal{O}(d)}$ in I and then use Lemma 2 to solve I in time $\delta^{2^{\mathcal{O}(d)}}(|I|)^{\mathcal{O}(1)}$. ◀

We would like to mention a corollary of Theorem 19 that we can derive very similarly to Corollary 20. Consider the #CSP problem, which asks for the number of satisfying assignments. A constraint language is *#-tractable* if #CSP is solvable in polynomial time for instances from $\mathcal{C}_\Gamma$ [2]. The proof of Lemma 2 can easily be adapted to #CSP, as at a variable node, we have to add, and at a component node we have to multiply. Hence, we can substitute in the statement of Corollary 20 CSP with #CSP and tractable with #-tractable.

7 Conclusion

In this work, we compute backdoors of bounded depth for the CSP to base classes defined via finite semi-conservative constraint languages. Our approach via obstruction trees seems to be fundamentally limited to semi-conservative languages. However, we are optimistic that our techniques can be extended to base classes of unbounded arity. A first step in this direction has already been obtained in the context of SAT for the base class of Horn formulas [14]. In this setting, it is particularly interesting to consider tractable classes (of unbounded arity) of CSPs based on restrictions on the graphical structure [7, 21, 22], as well as hybrid restrictions [8, 9, 11].

Another interesting direction for future research, which has also been mentioned in the context of SAT [14], are the so-called scattered and heterogeneous extensions of (strong) backdoor sets [19, 20].

These extensions can be readily lifted to backdoor depth by allowing each component to be in any of a given set of (heterogeneous) tractable base classes. Interestingly, while those two notions lead to orthogonal tractable classes in the context of backdoor size, they lead to the same notion for backdoor depth. Therefore, lifting these two extensions to backdoor depth, would result in a unified and significantly more general approach. Moreover, we think that obtaining a heterogeneous version of backdoor depth seems to be particularly promising within the context of CSP. This is because, in contrast to SAT, there is a wide range of tractable classes (even of bounded arity) that can be characterized in a unified manner via algebraic properties.

References

1 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24:1–24:66, 2011. doi:10.1145/1970398.1970400.

2 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. of the ACM*, 60(5):34:1–34:41, 2013. doi:10.1145/2528400.

3 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.

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

5 Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, 2016.

6 David Cohen and Peter Jeavons. The complexity of constraint languages. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume I, chapter 8, pages 245–280. Elsevier, 2006.

7 David Cohen, Peter Jeavons, and Marc Gyssens. A unified theory of structural tractability for constraint satisfaction and spread cut decomposition. In *International Joint Conferences on Artificial Intelligence (IJCAI-05)*, pages 72–77, 2005.

8 David A. Cohen, Martin C. Cooper, Páidí Creed, Dániel Marx, and András Z. Salamon. The tractability of CSP classes defined by forbidden patterns. *J. Artif. Intell. Res.*, 45:47–78, 2012.

9 David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Stanislav Zivný. Tractable classes of binary CSPs defined by excluded topological minors. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1945–1951. AAAI Press, 2015.

10 Martin C. Cooper, David A. Cohen, and Peter Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65(2):347–361, 1994. doi:10.1016/0004-3702(94)90021-3.

11 Martin C. Cooper, Philippe Jégou, and Cyril Terrioux. A microstructure-based family of tractable classes for CSPs. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 74–88. Springer Verlag, 2015.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.

14 Jan Dreier, Sebastian Ordyniak, and Stefan Szeider. SAT backdoors: Depth beats size. In *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. to appear. arXiv:2202.08326.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.

16 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.

17 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1670–1681. SIAM, 2016.

18 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.

19 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.

20 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.

21 Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

22 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.

23 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.

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

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

26 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006.

27 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.

28 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.

29 Marko Samer and Stefan Szeider. Fixed-parameter tractability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability, 2nd Edition*, chapter 17, pages 693–736. IOS Press, 2021. doi:10.3233/FAIA201000.

30 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.

31 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.

32 Ryan Williams, Carla Gomes, and Bart Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Informal Proc. of the Sixth International Conference on Theory and Applications of Satisfiability Testing, S. Margherita Ligure - Portofino, Italy, May 5-8, 2003 (SAT 2003)*, pages 222–230, 2003.

33 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.