The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)

Faster Algorithms for Weak Backdoors

Serge Gaspers, Andrew Kaploun

School of Computer Science and Engineering, UNSW Sydney
serge.gaspers@unsw.edu.au, andrewkaploun0@gmail.com

Abstract

A weak backdoor, or simply a backdoor, for a Boolean SAT formula ϕ into a class of SAT formulae $\mathcal{C}$ is a partial truth assignment τ such that $\phi[\tau] \in \mathcal{C}$ and satisfiability is preserved. The problem of finding a backdoor from class $\mathcal{C}_1$ into class $\mathcal{C}_2$, or WB($\mathcal{C}_1$,$\mathcal{C}_2$), can be stated as follows: Given a formula $\phi \in \mathcal{C}_1$, and a natural number k, determine whether there exists a backdoor for ϕ into $\mathcal{C}_2$ assigning at most k variables.

The class 0-VAL contains all Boolean formulae with at least one negative literal in each clause. We design a new algorithm for WB(3CNF, 0-VAL) by reducing it to a local search variant of 3-SAT. We show that our algorithm runs in time $O^*(2.562^k)$, improving on the previous state-of-the-art of $O^*(2.85^k)$. Here, the O^* notation is a variant of the big-O notation that allows to omit polynomial factors in the input size.

Next, we look at WB(3CNF, NULL), where NULL is the class consisting of the empty formula. This problem was known to have a trivial running time upper bound of $O^*(6^k)$ and can easily be solved in $O^*(3^k)$ time. We use a reduction to CONFLICT FREE d-HITTING SET to prove an upper bound of $O^*(2.2738^k)$, and also prove a lower bound of $2^{o(k)}$ assuming the Exponential Time Hypothesis.

Finally, HORN is the class of formulae with at most one positive literal per clause. We improve the previous $O^*(4.54^k)$ running time for WB(3CNF, HORN) problem to $O^*(4.17^k)$, by exploiting the structure of the SAT instance to give a novel proof of the non-existence of the slowest cases after a slight restructuring of the branching priorities.

Introduction

Boolean Satisfiability

Boolean Satisfiability, or *SAT*, is a foundational problem in computer science, serves as the canonical NP-Complete problem (Cook 1971), and solutions are useful in not only broader theoretical (Aho and Hopcroft 1974) and applied (Marques-Silva 2008) computer science, but a number of fields of AI (Järvisalo 2016). Noting that the fastest running time bound for 3-SAT algorithms is $O(1.307^n)$ (Hansen et al. 2019) (a bound which would imply solving a formula with just hundreds of variables is intractable), a discrepancy can been observed with modern SAT solving programs that can solve instances with hundreds of thousands of variables (Ohrimenko, Stuckey, and Codish 2007). Thus, it can be productive to formally investigate the structure of classes of SAT instances that can be solved faster than the general case. Throughout this paper, we will introduce how weak backdoor algorithms facilitate this investigation, explain their applications, and describe our improved algorithms.

The SAT problem consists of a Boolean formula ϕ; an expression consisting of variables with the binary operators OR ($\vee$) and AND ($\wedge$), the negation operator ($\neg$), and brackets. SAT asks whether there is an assignment of true (or 1) and false (or 0) values to its variables such that ϕ is satisfied. Defining a *literal* as a Boolean variable with or without a negation (e.g. $x, \neg y$); a *disjunctive clause*, or simply a *clause*, is a disjunctive ($\vee$) collection of literals (e.g. $(\neg x \vee y \vee x)$). A Boolean formula ϕ in *conjunctive normal form*, or *CNF*, consists of a conjunction ($\wedge$) of clauses (e.g. $C_1 \wedge ... \wedge C_n$ where $C_1, ..., C_n$ are clauses), and is in 3CNF if each clause has at most 3 variables. *3CNF-SAT*, or simply *3-SAT*, is the variant of SAT where $\phi \in$ 3CNF.

Parameterized Complexity

A parameterized problem is *fixed-parameter tractable* if it is solved by an algorithm that runs in $O(f(k) \cdot poly(n))$ time, where n is the input size, $poly(n)$ is a polynomial function of n, k is the parameter, and $f(k)$ is a computable function of the parameter only. In this way, a problem can be NP-hard but remain tractable so long as k is small, as we will illustrate when we discuss algorithms for backdoors. As usual in parameterized algorithmics (Cygan et al. 2015; Downey and Fellows 2013; Flum and Grohe 2006; Niedermeier 2006), we say that an algorithm runs in $O^*(f(k))$ time for parameter k if it runs in $O(f(k) \cdot poly(n))$ time.

Backdoors

Some Boolean satisfiability instances are easy to solve; others seem very difficult to solve. Backdoors formalize this phenomenon.

Williams, Gomes, and Selman (2003) introduced the idea of SAT backdoors as a way of formalising the intuitive structure found within a lot of SAT formulae. A (*partial*) *assignment* τ is a mapping from variables to truth values. If we have a formula ϕ and an assignment τ, we denote ϕ with

τ applied to it by $\phi[\tau]$, which denotes the formula obtained from ϕ by removing all clauses that contain a true literal under τ and by removing all false literals from the remaining clauses. We will say that a *weak backdoor assignment* from class $\mathcal{C}_1$ to $\mathcal{C}_2$ for a formula $\phi \in \mathcal{C}_1$ is a truth assignment $\tau : S \to \{\mathsf{true}, \mathsf{false}\}$ to a subset of variables $S \subseteq Var(\phi)$ such that $\phi[\tau] \in \mathcal{C}_2$, and $\phi[\tau]$ is satisfiable. We call a *weak backdoor set* a set of variables S to which there exists an assignment of values that is a weak backdoor assignment. We also call weak backdoor sets *weak backdoors*, *backdoors*, or simply *WB*. Given an input parameter k and a class $\mathcal{C}_2$, the weak backdoor problem for classes $\mathcal{C}_1, \mathcal{C}_2$, or WB($\mathcal{C}_1, \mathcal{C}_2$), asks us to find a backdoor to $\mathcal{C}_2$ of no more than k variables for a formula $\phi \in \mathcal{C}_1$ (Gaspers and Szeider 2012).

How can this be applied to solve SAT instances? Take some class of SAT instances $\mathcal{C}$, for instance 2CNF, for which the satisfiability problem is solvable in polynomial time. Although such instances would of course be exceedingly rare in the wild, instances that are 'close' to a desired class are more common, where a formula ϕ is 'close' if there exists a small backdoor from ϕ to $\mathcal{C}$. Then, if we have a fixed-parameter tractable algorithm for WB(3CNF, $\mathcal{C}$), so long as the backdoor is small we can find a backdoor reasonably quickly; and then determine if the resulting formula in $\mathcal{C}$ is satisfiable in polynomial time. To show how backdoors formalize insights into practical algorithms, observe that DPLL-style (Davis, Logemann, and Loveland 1962) branching algorithms for SAT implicitly find backdoors: When a satisfying assignment is found in a leaf of the search tree, the path from the root to this leaf corresponds to a weak backdoor. Empirically, Li and van Beek (2011) have investigated the sizes of backdoors of large real-world instances, and found that size of the backdoor was usually less than 0.5% of the number of variables in the formula.

KROM First, note that KROM is synonymous with 2CNF. Misra et al. (2013) proved a bound of $O^*(2.0755^k)$ for WB(3CNF, KROM) by reducing it to a problem called 3-HITTING SET.

Given a collection of sets $\mathcal{C}$, each containing elements from a universe $\mathcal{U}$, a *hitting set* is a subset $S \subseteq \mathcal{U}$ such that S has nonempty intersection with every set in $\mathcal{C}$, that is, for each set $C \in \mathcal{C}$, $C \cap S \neq \emptyset$. The d-HITTING SET problem gives a collection $\mathcal{C}$ of subsets that all have cardinality at most d, and a parameter k, and asks whether there exists a hitting set of size at most k. We can sketch the reduction by Misra et al. (2013) by taking a formula ϕ, turning clauses into subsets, and turning every unique literal into a unique element. Then, noting that a backdoor from 3CNF to KROM simply has to 'hit' one element in each clause, a backdoor for ϕ would map onto a hitting set.

Jain, Kanesh, and Misra (2020) created a variant called CONFLICT FREE d-HITTING SET, consisting of a collection $\mathcal{C}$, a universe $\mathcal{U}$, and additionally a conflict graph G where each vertex corresponds uniquely to an item in $\mathcal{U}$, and integer k; and asks whether there exists a hitting set S of size at most k, such that the induced subgraph $G[S]$ has no edges. Jain, Kanesh, and Misra (2020) created an $O^*(2.2738^k)$ algorithm for this. We will find that similarly to 3-HITTING-SET and WB(3CNF, KROM), this problem closely relates to WB(3CNF, NULL).

HORN HORN is the class of formulae where each clause has at most one positive literal. The problem of finding a weak backdoor from 3CNF to HORN was first discussed in (Misra et al. 2013), where a bound of $O^*(4.54^k)$ was proven using a branching algorithm. HORN has applications in model theory, automata, and automatic theorem proving (Moore et al. 2005) (Makowsky 1987).

0-VAL 0-VAL is the class of SAT formulae where each clause contains at least one negative literal. Note a fundamental property of 0-VAL, that an all-false assignment satisfies any such formula. (Raman and Shankar 2013) proved an $O^*(2.85^k)$ bound for the equivalent form of this problem; MINIMUM WEIGHT 3-SAT, which asks whether a 3-SAT formula ϕ can be satisfied by assigning no more than k variables true (and all other variables false). They noted the equivalence by recalling the aforementioned fundamental property, and thus that if there is a backdoor into 0-VAL with k truevariables, then setting every other variable in ϕ to falsewill satisfy ϕ, giving an assignment of weight no more than k. We will use this intuition to guide Lemma 8.

NULL NULL is the class consisting of only the Null formula (the formula with no clauses, that is trivially satisfied). When Raman and Shankar (2013) proved a $O^*(2.85^k)$ bound for WB(3CNF, 0-VAL), they mentioned in their conclusion that a possible future research direction is to find a parameterized algorithm for WB(3CNF, NULL), and recognised the trivial bound of $O^*(6^k)$. One can sketch a recursive $O^*(3^k)$ algorithm in the following way: *If $\phi \notin$ NULL, pick an arbitrary clause C. For each literal in C, recurse into a state where we add an assignment to τ that makes the literal* true*. If we have more than k variables in τ, do not recurse any further. If $\phi \in$ NULL, then there is a backdoor of size no more than k.* Then, since at each node of the recursion tree we branch into 3 nodes, and the depth is no more than k, the total number of leaves in the search tree is bounded by 3^k, and since the time spent in each node is polynomial in $n \geqslant k$, the running time is $O^*(3^k)$.

Bounds Previously, the best known O^* running time bounds for finding backdoors from 3CNF to a class $\mathcal{C}$ were:

$\mathcal{C}$	HORN	KROM	0-VAL	NULL
Upper Bound	4.54^k	2.0755^k	2.85^k	6^k
Lower Bound	2^k	2^k	$2^{o(k)}$	None

Table 1: Summary of Previous Bounds

The 2^k lower bounds are subject to the Strong Exponential Time Hypothesis, and the $2^{o(k)}$ lower bounds are subject to the Exponential Time Hypothesis. The upper bounds for 0-VAL and NULL were given by Raman and Shankar (2013). The other bounds in the table were proved by Misra et al. (2013). Since their bound for KROM was bound by

the running time for 3-HITTING-SET, we update the upper bound to reflect the new result proved by Wahlström (2007).

The new bounds, with our improvements in **bold**, are:

$\mathcal{C}$	HORN	KROM	0-VAL	NULL
Upper Bound	$\mathbf{4.17^k}$	2.0755^k	$\mathbf{2.562^k}$	$\mathbf{2.2738^k}$
Lower Bound	2^k	2^k	$2^{o(k)}$	$\mathbf{2^{o(k)}}$

Table 2: Summary of New Bounds

Preliminaries

The Exponential Time Hypothesis

The *Exponential Time Hypothesis*, or *ETH*, postulates that there is no algorithm that decides the satisfiability of a CNF formula with n variables in time $O^*(2^{o(n)})$ (Impagliazzo and Paturi 2001; Impagliazzo, Paturi, and Zane 2001).

Ball-3-SAT

One approach that has been used in the literature to solve SAT quickly has been to 'cover' the solution space with a series of local searches. Represent an assignment to the set of variables $x_1, x_2, ..., x_n$ by a *bitstring* of length n, that is, an element of the set $\{0,1\}^n$, also known as the Hamming space. This will be the space that we wish to 'cover'. Then, define the *Hamming distance* between two equal length bitstrings as the number of positions in which they differ. The ball in the Hamming space, $B_{\mathcal{H}}(s,r)$, denotes the set of all bitstrings no more than distance r from string s.

A *covering code* $\mathcal{C}$ is a set of bitstrings such that $\bigcup_{c\in\mathcal{C}} B_{\mathcal{H}}(c,r) = \{0,1\}^n$. At this point, we can observe that if our covering code had $O^*(f(k))$ elements, and we could determine if there existed a satisfying assignment/bitstring $b \in B_{\mathcal{H}}(s,r)$ for ϕ in running time $O^*(g(k))$, then we could solve SAT in $O^*(f(k)\cdot g(k))$. And indeed, many state of the art algorithms for 3-SAT have involved this process (see, e.g., Brueggemann and Kern (2004); Scheder (2008); Kutzkov and Scheder (2010)).

This motivates the object of our interest, the BALL-3-SAT problem, which asks whether there exists a satisfying assignment for a formula ϕ within Hamming distance r of α (Fomin and Kratsch 2010). The bound given by Kutzkov and Scheder (2010) will come in handy for proving our bound for WB(3CNF, 0-VAL).

Branching Algorithms

In our algorithm for WB(3CNF, HORN), we use a branching algorithm (Fomin and Kratsch 2010) where in each Branching Rule we branch into n cases that add $b_1, b_2, ...,$ and b_n variables to the assignment, summarized by the *branching vector* $(b_1, b_2, ..., b_n)$. Then, the number of leaves in the branching tree for that Branching Rule is bound by the function $T(k) \leqslant T(k-b_1) + ... + T(k-b_n)$, and the unique positive real root of $x^k = x^{k-b_1} + ... + x^{k-b_n}$ is the *branching number* $\mathcal{B}$, hence the running time of our algorithm is $O^*(B^k)$, where B is the maximum branching number of any branching rule (Cygan et al. 2015; Fomin and Kratsch 2010).

WB(3CNF, HORN)

Algorithm & Analysis

Inspired by Misra et al. (2013), we define a *C2* clause as a 3-clause with exactly two positive literals. (*e.g.* $(\neg a, b, c)$). Rules adapted from that paper are marked with a ($\star$).

Apply the following branching rules exhaustively:

Branching Rule 1.($\star$) If there is a positive 3-clause (x,y,z), branch into the following cases:

- $x = 1$
- $y = 1$
- $z = 1$
- $x = 0, y = 0$
- $x = 0, z = 0$
- $y = 0, z = 0$

Branching Rule 2.($\star$) If there is a positive 2-clause (x,y), branch into

- $x = 1$
- $x = 0$
- $y = 1$
- $y = 0$

Branching Rule 3.($\star$) If variable x occurs positively in both $C = (x,y,\neg z)$ and $C' = (x,y',\neg z')$, branch into

- $x = \{1,0\}$
- $y = \{1,0\}, y' = \{1,0\}$
- $y = \{1,0\}, z' = 0$
- $z = 0, y' = \{1,0\}$
- $z = 0, z' = 0$

Here, by a notation such as $y = \{1,0\}, y' = \{1,0\}$ we mean all 4 branches where y is set to one value among $\{1,0\}$ and y' is set to one value among $\{1,0\}$.

In Branching Rule 4, suppose two clauses C and C' have exactly two variables x, y in common.

Branching Rule 4.1.($\star$) If x is positive in C and C' and y is negative in C and C', then we have $C = (x,\neg y,z), C' = (x,\neg y,z')$. Branch into

- $x = \{1,0\}$
- $y = 0$
- $z = \{1,0\}, z' = \{1,0\}$

Branching Rule 4.2.($\star$) If x is positive in C and negative in C' and y is negative in C and positive in C', then we have $C = (x,\neg y,z), C' = (\neg x,y,z')$. Branch into

- $x = 0$
- $y = 0$
- $x = 1, y = 1$
- $x = 1, z' = \{1,0\}$
- $y = 1, z = \{1,0\}$
- $z = \{1,0\}, z' = \{1,0\}$

Branching Rule 4.3. If x is positive in C and C', and y is positive and negative in C and C', respectively, then we have $C = (x, y, \neg z), C' = (x, \neg y, z')$. Branch into

- $x = 1$
- $y = 1$
- $y = 0$
- $y = 1, z' = \{1, 0\}$
- $z = 0, z' = \{1, 0\}$

In Branching Rule 5, we branch on C2 clauses that contain a variable that does not occur in any other C2 clauses.

Branching Rule 5.1. If we have a C2 clause containing a variable x that only occurs once in a C2 clause, and the occurrence is as a negative literal, we have $(\neg x, y, z)$, in which case we branch into

- $y = 1$
- $y = 0$
- $z = 1$
- $z = 0$

Branching Rule 5.2. If we have a C2 clause containing a variable x that only occurs once in a C2 clause, and the occurrence is as a positive literal, we have $(x, \neg y, z)$, in which case we branch into

- $y = 0$
- $z = 1$
- $z = 0$

In Branching Rule 6, Suppose C, C' are C2 clauses with x, y both positive in C, C'.

Branching Rule 6.1. Suppose x and y occur outside of C, C', and one of these occurrences is within the same clause. Thus we have $C = (x, y, \neg z)$, $C' = (x, y, \neg z')$ and $C'' = (x, y, \neg z'')$. In this case, we can branch into

- $x = \{1, 0\}$
- $y = \{1, 0\}$
- $z = 0, z' = 0, z'' = 0$

Branching Rule 6.2. Suppose we have $(\neg x, v_0, w_0)$ and $(\neg y, v_1, w_1)$ where pairs v_i, v_j and w_i, w_j are not necessarily distinct, but pairs v_i, w_i are necessarily distinct. Then, to make $C = (x, y, \neg z), C' = (x, y, \neg z')$ HORN, branch into

- $x = 1, v_0 = \{1, 0\}$
- $x = 1, w_0 = \{1, 0\}$
- $x = 0$
- $y = 1, v_1 = \{1, 0\}$
- $y = 1, w_1 = \{1, 0\}$
- $y = 0$
- $z = 0, z' = 0$

Branching Rule 6.3. If x and y do not occur in any other C2 clauses, branch into

- $x = 1$
- $x = 0$
- $z = 0, z' = 0$

In Branching Rule 7, suppose C2 clauses C, C' have 3 variables in common. That is, $C = (x, y, \neg z)$ and $C' = (x, \neg y, z)$.

Branching Rule 7.1. Suppose there exists a clause $C'' = (\neg x, y, z)$, and y, z occur in no other C2 clauses. Then, branch into

- $x = 0, y = 0$
- $x = 0, y = 1$
- $x = 1, y = 0$
- $x = 1, y = 1$

Branching Rule 7.2. If both y and z occur in no other C2 clauses, we can simply branch into

- $x = 1$
- $x = 0$

In order to prove that all of our branching rules are exhaustive, we will introduce the *Branching Exhaustion Lemma*.

Definition 1. *A* branching *is a set of assignments* $\{\tau_1, \tau_2, ...\}$ *that we branch into in our branching algorithm.*

Definition 2. *A branching is* exhaustive *if at least one of the branches results in a backdoor of minimum size, assuming a backdoor exists.*

Rule	Branching Vector	Branching Number
1	$(1, 1, 1, 2, 2, 2)$	3.80
2	$(1, 1, 1, 1)$	4
3	$(1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2)$	4.17
4.1	$(1, 1, 1, 2, 2, 2, 2)$	4
4.2	$(1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2)$	4.17
4.3	$(1, 1, 1, 2, 2, 2, 2)$	4
5.1	$(1, 1, 1, 1)$	4
5.2	$(1, 1, 1)$	3
6.1	$(1, 1, 1, 1, 3)$	4.07
6.2	$(1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2)$	4.17
6.3	$(1, 1, 2)$	2.42
7.1	$(2, 2, 2, 2)$	2
7.2	$(1, 1)$	2

Table 3: Branching Vectors and Branching Numbers for WB(3CNF, HORN)

Lemma 1 (The Branching Exhaustion Lemma)**.** *For any set of clauses S, if a branching consists of every minimal assignment that makes all clauses in S either satisfied or in the* base class*, $\mathcal{C}_2$, then the branching is exhaustive.*

Proof. Clearly, every backdoor for ϕ must contain at least one of the backdoors for S inside it. □

Lemma 2. *Every branching rule is exhaustive.*

Proof. For Branching Rule 5.1 and Branching Rule 5.2, suppose there exists a weak backdoor $\tau \ni x$ of size at most k. Then since x occurs in no C2 clauses outside of C , if we replace the assignment of x in the backdoor with either $z = \mathsf{true}$ or $z = \mathsf{false}$, τ will still be a backdoor and the equation will remain satisfiable for at least one of the choices. Thus we do not need to branch on assigning x. For Branching Rule 6.3, suppose there exists a weak backdoor $\tau \ni y$ of size at most k. Then since y occurs in no C2 clauses outside of C, C' , if we replace the assignment of y in the backdoor with either $x = \mathsf{true}$ or $x = \mathsf{false}$, τ will still be a backdoor and the equation will remain satisfiable for at least one of the choices. Thus we do not need to branch on assigning y. For Branching Rule 7.1, clearly, we need to assign at least two variables to make C, C', and C'' HORN. Since x, y, and z only occur in C2 clauses C, C', C'', we can simply branch on the possible assignments of x and y since it is impossible to make the backdoor smaller by putting z in the backdoor instead. For Branching Rule 7.2, suppose there exists a weak backdoor $\tau \ni y$ (or WLOG z) of size at most k. Then since y occurs in no C2 clauses outside of C, C' , if we replace the assignment of y in the backdoor with either $x = \mathsf{true}$ or $x = \mathsf{false}$, τ will still be a backdoor and the equation will remain satisfiable for at least one of the choices. Thus we do not need to branch on assigning y. The exhaustiveness of all other rules follows immediately from the Branching Exhaustion Lemma. □

The main novelty of our algorithm is a case analysis that avoids several branching rules with high branching numbers. To show the case analysis is exhaustive, assume, for the sake of contradiction, there exists a formula ϕ that is not Horn and to which none of the branching rules applies.

Since Branching Rule 5 eliminates all C2 clauses with no shared variables with other clauses, ϕ contains C2 clauses that share at least one variable. However, we will now show that no two C2 clauses share two variables, no two C2 clauses share one variable, and then that no two C2 clauses share three variables, giving a contradiction.

Lemma 3. *The formula ϕ contains no two C2 clauses sharing exactly two variables.*

Proof. In Branching Rule 4 we eliminate all C2 clauses that share 2 variables with another clause, except the subcases where x, y both occur positively in two clauses C, C', which we address in Branching Rule 6. To see why Branching Rule 6 is exhaustive, observe the following: if x or y occurred elsewhere positively (individually) in different clauses, this would have been branched on in Branching Rule 3.1. If they occurred elsewhere in the same clause and either x or y was negative, this would be branched on in Branching Rule 4. □

At this point, we note that Lemma 4 formalizes the main conceptual contribution of our algorithm, hence we give some intuition to aid with understanding the proof. Recall that we wish to exhaustively branch on every case where a non-HORN instance can occur. At first glance, if we consider the clauses $C = (\neg x, y, z), C' = (\neg x, y', z')$, it appears we have failed to exhaust every case in our analysis. However, we will prove that this set of clauses cannot exist, as follows: Suppose we do have these two clauses. Since variables that occur in only one C2 clause have been eliminated, consider another clause that y' occurs in. Upon close examination, we can see that y' cannot occur in the same C2 clause as x, y, z, or z', since we would have branched on all of the possible cases, so ϕ must have a clause $(\neg y, a, b)$. Similarly, upon a close examination of a, we can see that a must also occur in another C2 clause, which cannot also contain x, y, z, y', or b. Hence, ϕ must have the clause $(\neg a, c, d)$. At this point, one may notice a pattern, and suspect that if we have applied all of our branching rules, we can can go down an infinite regress of clauses that must exist in ϕ. We will proceed to formalize this intuitive notion to show that we exhaustively branch on any HORN clauses in ϕ.

Our proof that no two C2 clauses in ϕ share one variable is based on the following definitions.

Definition 3. *A variable is a* child *with respect to a set of clauses S if it occurs only positively in S.*

Definition 4. *A clause is a* child *with respect to a set of clauses S if it is a C2 clause containing at least one variable that is a child with respect to S.*

Definition 5. *A set S of C2 clauses is a z-*branching *if no variables except $\neg z$ occur only negatively in S, and it contains at least 2 child clauses with respect to itself.*

Lemma 4. *The formula ϕ has no pair of C2 clauses C, C' sharing exactly one variable x, that is negative in C and positive in C'.*

Proof. Suppose that the previous branching rules have been exhaustively applied, and the instance I contains C2 clauses $C = (\neg x, y, z)$, $C' = (x, y', \neg z')$ with exactly one variable x in common, where x is negative in C and positive in C'. Since I trivially contains a z'-branching, there exists a largest z'-branching $S \subseteq I$. Then consider an arbitrary child clause $(\neg a, b, c) \in S$, where c may or may not be equal to z'. Due to Branching Rule 3 and Lemma 3, we must have a C2 clause $C_I \in I \setminus S$ containing $\neg b$, alongside two variables which do not exist in S. (Note that we cannot have $C_I = (\neg b, a, c)$ since a occurs positively elsewhere in S by Definition 9.) Thus, $S \cup \{C_I\}$ is a branching set that contradicts maximality. □

Lemma 5. *The formula ϕ has no two C2 clauses C, C' sharing exactly one variable x, that is negative in C and in C'.*

Proof. Suppose x occurs negatively in $C = (\neg x, y, z), C' = (\neg x, y', z')$. Then, by exhaustive application of Branching Rule 5, we must have y occurring

elsewhere, and due to Branching Rule 3 and Branching Rule 4, this occurrence must be negative and not involving x or z (noting that we cannot have $(\neg y, x, z)$ because then x would appear negatively in one clause and positively in another). But then, y occurs positively in one clause and negatively in another clause, contradicting Lemma 4. □

Lemma 6. *The formula ϕ contains no two C2 clauses sharing exactly one variable x.*

Proof. By Branching Rule 3, there are no pairs of C2 clauses C, C' such that x is positive in C and positive in C'. Due to Lemma 4, there are no pairs of C2 clauses C, C' such that x is negative in C and positive in C'. Lemma 5 shows there are no pairs of C2 clauses C, C' such that x is negative in C and in C'. This exhausts the combinations of C2 clauses for the lemma. □

Lemma 7. *The formula ϕ has no two C2 clauses sharing 3 variables.*

Proof. After Branching Rule 7 has been exhausted, clearly, z must exist outside of C, C', and such that $z \in C''$ and C'' does not consist only of variables x, y, z. If two of the variables occurred elsewhere together with a negative occurrence, this would contradict Lemma 3, and if they occurred elsewhere separately, this would contradict Lemma 6. □

This proves the correctness of our algorithm. Taking the maximum of all of the branching numbers to bound the number of nodes in the search tree, we obtain

Theorem 1. *WB(3CNF, HORN) can be solved in $O^*(4.17^k)$ time.*

WB(3CNF, 0-VAL)

Algorithm & Analysis

First, we prove a lemma relating WB(3CNF, 0-VAL) to BALL-3-SAT.

Lemma 8. *A formula ϕ has a 0-VAL backdoor of size at most k if and only if it has an assignment τ of size at most k that only assigns* true *such that $\phi[\tau] \in$ 0-VAL.*

Proof. Since the backwards direction is trivial, consider only the forward implication, and suppose a backdoor of size at most k exists. Then, there is an assignment τ for this backdoor such that $\phi[\tau] \in$ 0-VAL. Let τ' be τ, but restricted to only its true assignments. Since $\phi[\tau]$ is satisfied by the all-false assignment, then $\phi[\tau']$ must also be satisfied by the all-false assignment, and hence $\phi[\tau'] \in$ 0-VAL. □

Now, recall the BALL-3-SAT algorithm from Kutzkov and Scheder (2010) that runs in $O^*(2.562^r)$ time, where r is the maximum Hamming distance from the starting assignment. Our algorithm involves a reduction to BALL-3-SAT.

Theorem 2. *There is an algorithm for WB(3CNF, 0-VAL) that runs in $O^*(2.562^k)$ time.*

Proof. Consider the following algorithm for WB(3CNF, 0-VAL): Given a formula ϕ and parameter k, run the algorithm for Ball-3-SAT on ϕ, with maximum distance k and a starting assignment of only false. If Ball-3-SAT returns a satisfying assignment, return a backdoor with every true variable in the assignment; otherwise, return that there is no backdoor.

This algorithm clearly has a running time bound of $O^*(2.562^k)$ since the only exponential stage is running the Ball-3-SAT algorithm. The algorithm's correctness follows from the fact that by Lemma 8, a 0-VAL backdoor of size no more than k exists if and only if an assignment of size no more than k exists that only has truth values; then this assignment is an assignment whose distance from the all-false assignment is no more than k. □

WB(3CNF, NULL)

Lower Bound

First, we can trivially prove a lower bound for problem, conditional on the Exponential Time Hypothesis.

Theorem 3. *There is no algorithm that solves WB(3CNF, NULL) in $2^{o(k)}$ time, assuming the ETH.*

Proof. Recall that our algorithm outputs a weak backdoor set of size no more than k if one exists; otherwise it outputs that no such backdoor exists. Suppose that there exists an algorithm that solves WB(3CNF, NULL) in $O^*(2^{o(k)})$ time. Then, the following procedure solves 3-SAT: Given any formula ϕ, run the algorithm with $k = n$, where n is the number of variables in ϕ, and return Yes if the algorithm returns a backdoor, and No otherwise. Thus, we can solve 3-SAT in $O^*(2^{o(n)})$, contradicting ETH. □

Algorithm & Analysis

Theorem 4. *There is an algorithm for WB(3CNF, NULL) that runs in $O^*(2.2738^k)$ time.*

Proof. We will construct a reduction from WB(3CNF, NULL) to CONFLICT FREE d-HITTING SET. Consider an instance of WB(3CNF, NULL) with formula ϕ. The universe $\mathcal{U}$ will contain the elements x, x' for all variables $x \in \phi$, and similarly the conflict graph G will consist of the edges (x, x') for all variables $x \in \phi$. For our construction, denoting the literals in ϕ by $Lit(\phi)$, we will use the function $f : Lit(\phi) \to \mathcal{U}$

$$f(l) = \begin{cases} x & \text{if } l = x \\ x' & \text{if } l = \neg x \end{cases}$$

Then, let $\mathcal{C}$ consist of $\bigcup_{l \in C} f(l)$ for each clause $C \in \phi$.

We now show that this is a reduction. Suppose that there exists a conflict-free hitting set H of size no more than k for $(\mathcal{C}, G)$. Then, denoting $var(l)$ to be the variable underlying the literal l, consider the set $\bigcup_{l \in H} f^{-1}(l)$. This must be a backdoor to ϕ, since every clause has a true literal, and we do not assign any variable both true and false, since this is excluded by the conflict graph. Conversely, suppose there is a backdoor set for ϕ of size no more than k, and correspondingly a weak backdoor assignment τ. To construct a hitting set for $\mathcal{C}$, for each variable $x \in \tau$, if it is assigned true, add

x, and if it assigned false, add x'. Then, clearly it has size no more than k, and cannot contain both x and x' for any variable x; thus it is conflict-free. □

Conclusion

We improve on a number of algorithms to find weak backdoors of 3-CNF formulae. Along the way, we observe some correspondences between backdoor problems seen in the literature and known combinatorial problems, and contribute new ideas for solving problems in which we can prove that certain structures (that may intuitively seem to be present) do not exist.

We discussed ways to construct a SAT sub-solver with backdoor algorithms that are of theoretical interest: If a SAT instance contains a small backdoor into one of these classes, we can use a backdoor algorithm to find it, and then solve the rest of the instance in polynomial time. Some have empirically investigated the size of these backdoors in practical instances, and the observations made in our algorithms could help to improve real-world algorithms for finding these backdoors.

To improve our algorithm for WB(3CNF, HORN), naively attempting a finer case distinction may be problematic since one may have to consider cases that contain undesirable structures like $C = (\neg x, y, z)$, $C' = (x, y', \neg z')$. However, after we branch, we do not look at the new clauses that are created, even though it is possible some of these could give us a more favorable branching. An improvement could be to track the number of clauses that exist that give a lower branching number than 4.17; and then apply Measure and Conquer to these clauses, as used for 3-HITTING SET by Niedermeier (2006), Fernau (2010), and Wahlström (2007), to bound the overall running time.

Acknowledgments

We thank an anonymous reviewer from another conference for suggesting the reduction to CONFLICT FREE d-HITTING SET.

References

Aho, A. V.; and Hopcroft, J. E. 1974. *The Design and Analysis of Computer Algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc., 1st edition. ISBN 0201000296.

Brueggemann, T.; and Kern, W. 2004. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science*, 329(1): 303 – 313.

Cook, S. A. 1971. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, 151–158. New York, NY, USA: Association for Computing Machinery. ISBN 9781450374644.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer. ISBN 978-3-319-21274-6.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A Machine Program for Theorem-Proving. *Commun. ACM*, 5(7): 394–397.

Downey, R. G.; and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer. ISBN 978-1-4471-5558-4.

Fernau, H. 2010. A Top-Down Approach to Search-Trees: Improved Algorithmics for 3-Hitting Set. *Algorithmica*, 57(1): 97–118.

Flum, J.; and Grohe, M. 2006. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer. ISBN 978-3-540-29952-3.

Fomin, F. V.; and Kratsch, D. 2010. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer. ISBN 978-3-642-16532-0.

Gaspers, S.; and Szeider, S. 2012. *Backdoors to Satisfaction*, 287–317. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-30891-8.

Hansen, T. D.; Kaplan, H.; Zamir, O.; and Zwick, U. 2019. Faster k-SAT Algorithms Using Biased-PPSZ. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, 578–589. New York, NY, USA: Association for Computing Machinery. ISBN 9781450367059.

Impagliazzo, R.; and Paturi, R. 2001. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2): 367–375.

Impagliazzo, R.; Paturi, R.; and Zane, F. 2001. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4): 512–530.

Jain, P.; Kanesh, L.; and Misra, P. 2020. Conflict Free Version of Covering Problems on Graphs: Classical and Parameterized. *Theory of Computing Systems*, 64(6): 1067–1093.

Järvisalo, M. 2016. Boolean Satifiability and Beyond: Algorithms, Analysis, and AI Applications. In Kambhampati, S., ed., *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 4066–4069. IJCAI/AAAI Press.

Kutzkov, K.; and Scheder, D. 2010. Using CSP To Improve Deterministic 3-SAT. *CoRR*, abs/1007.1166.

Li, Z.; and van Beek, P. 2011. Finding Small Backdoors in SAT Instances. In Butz, C.; and Lingras, P., eds., *Advances in Artificial Intelligence*, 269–280. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-21043-3.

Makowsky, J. 1987. Why horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34(2): 266 – 292.

Marques-Silva, J. 2008. Practical applications of boolean satisfiability. In *2008 9th International Workshop on Discrete Event Systems*, 74–80. IEEE.

Misra, N.; Ordyniak, S.; Raman, V.; and Szeider, S. 2013. Upper and Lower Bounds for Weak Backdoor Set Detection. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, 394–402. Springer.

Moore, C.; Istrate, G.; Demopoulos, D.; and Vardi, M. Y. 2005. A Continuous-Discontinuous Second-Order Transition in the Satisfiability of Random Horn-SAT Formulas. In Chekuri, C.; Jansen, K.; Rolim, J. D. P.; and Trevisan, L., eds., *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, 414–425. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-31874-3.

Niedermeier, R. 2006. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press. ISBN 9780198566076.

Ohrimenko, O.; Stuckey, P. J.; and Codish, M. 2007. Propagation = Lazy Clause Generation. In Bessière, C., ed., *Principles and Practice of Constraint Programming – CP 2007*, 544–558. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-74970-7.

Raman, V.; and Shankar, B. S. 2013. Improved Fixed-Parameter Algorithm for the Minimum Weight 3-SAT Problem. In *WALCOM: Algorithms and Computation, 7th International Workshop, WALCOM 2013, Kharagpur, India, February 14-16, 2013. Proceedings*, volume 7748 of *Lecture Notes in Computer Science*, 265–273. Springer.

Scheder, D. 2008. Guided Search and a Faster Deterministic Algorithm for 3-SAT. In Laber, E. S.; Bornstein, C. F.; Nogueira, L. T.; and Faria, L., eds., *LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings*, volume 4957 of *Lecture Notes in Computer Science*, 60–71. Springer.

Wahlström, M. 2007. *Algorithms, measures and upper bounds for satisfiability and related problems*. Ph.D. thesis, Linköping University, Sweden.

Williams, R.; Gomes, C. P.; and Selman, B. 2003. Backdoors To Typical Case Complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 1173–1178. Morgan Kaufmann.