Ann Math Artif Intell (2014) 70:399–431
DOI 10.1007/s10472-014-9407-9

Tradeoffs in the complexity of backdoors to satisfiability: dynamic sub-solvers and learning during search

Bistra Dilkina · Carla P. Gomes · Ashish Sabharwal

Published online: 15 March 2014

Abstract There has been considerable interest in the identification of structural properties of combinatorial problems that lead to efficient algorithms for solving them. Some forms of structure, such as properties of the underlying constraint graph, are "easily" identifiable. Others, such as backdoor sets, are of interest because they capture key aspects of state-of-the-art constraint solvers as well as of many real-world problem instances. While backdoors were originally introduced to capture propagation based simplification mechanisms of solvers, they have recently been studied also in the context of tractable syntactic classes such as 2CNF and Horn. These syntactic classes, however, do not capture key aspects of solvers such as empty clause (i.e., violated constraint) detection. We show that incorporating inconsequential sounding features such as empty clause detection has a dramatic impact on both the complexity of finding a backdoor of size k (which becomes harder in the worst case) and on the size of the resulting backdoor (which can become arbitrarily smaller). Empirically, we show that commonly employed polynomial-time "dynamic" constraint propagation mechanisms, such as unit propagation, pure literal elimination, and probing, often lead to much smaller backdoor sets in real-world domains than "statically" defined classes such as Horn and RHorn, thereby capturing structure much more succinctly. We also reveal the inherent limits of the simpler concept of deletion backdoors, specifically by looking at renamable Horn sub-formulas. Finally, we extend the notion of backdoors to incorporate learning during search—a key aspect of nearly all state-of-the-art systematic SAT solvers—and show, both theoretically and empirically, that this drastically reduces the size of the resulting backdoor set. Our results suggest that structural notions explored

B. Dilkina (✉)
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA
e-mail: bdilkina@cc.gatech.edu

C.P. Gomes
Department of Computer Science, Cornell University, Ithaca, NY 14853, USA
e-mail: gomes@cs.cornell.edu

A. Sabharwal
IBM Watson Research Center, Yorktown Heights, NY 10598, USA
e-mail: ashish.sabharwal@us.ibm.com

for designing efficient algorithms for combinatorial problems should capture both statically and dynamically identifiable properties of the combinatorial problem being solved.

Keywords Boolean satisfiability · Backdoor sets · Horn RHorn · Clause learning · Search

Mathematics Subject Classifications (2010) 68T20 · 03B05

1 Introduction

General purpose inference engines for Constraint Satisfaction Problems (CSPs), in particular for Boolean satisfiability (SAT), have witnessed tremendous progress since the early 1990s, and are increasingly becoming key components of efficient solution methodologies for many interesting problems within the realm of artificial intelligence as well as hardware and software verification. While a lot of the work done in this area in the late 1980s and early 1990s, especially on local search methods, was directed towards solving difficult combinatorial puzzles such as the N-queens problem or randomly generated problem instances that provided a wide range of challenging parameterized benchmarks to push the solvers, the development of successful techniques quickly led to the generation of a whole suite of non-random instances of direct interest to AI practitioners and the verification community alike. For example, work on bounded model checking using symbolic as well as SAT-based techniques [4, 6] and planning using SAT encodings [36, 37] demonstrated the tremendous potential of general purpose constraint reasoning engines as powerful tools for these communities.

Fortunately, while being able to efficiently solve all CSP or SAT instances with one polynomial-time algorithm is highly unlikely as these problems are NP-complete, many real-world instances have been found to exhibit "structural properties" very different from those of hard randomly generated instances, structures that modern constraint solvers are in fact able to exploit to, in a sense, defy the worst-case complexity of these problems. Specifically, while one would expect instances of an NP-hard problem with hundreds of thousands of variables to be completely out of reach in practice, it is not uncommon to be able to solve many such "structured" instances of interest within a few minutes to a few hours using state-of-the-art constraint solvers.

Capturing and exploiting problem structure is key to solving large real-world combinatorial problems. A very fruitful and prolific line of research that has been pursued in the study of combinatorial problems is the identification of various structural properties of instances that lead to efficient algorithms. Ideally, one prefers structural properties that are "easily" identifiable, such as topological properties of the underlying constraint graph. As an example, the degree of acyclicity of a constraint graph, measured using various graph width parameters, plays an important role with respect to the identification of tractable instances [8, 12, 13, 27–29, 54]. Other useful structural properties consider the nature of the constraints, such as their so-called functionality, monotonicity, and row convexity [15, 58].

Another approach for studying the nature of combinatorial problems of interest focuses on the role of *hidden structure*. One example of such hidden structure is a *backdoor set*—a set of variables B such that once they are *instantiated*, the remaining problem simplifies to a tractable class [59, 60]. The notion of simplification or tractability in the definition of backdoor sets is captured by a polynomial-time algorithm or sub-solver that, given a formula, either correctly decides its satisfiability or rejects it. This easily captures the behavior of the propagation procedures of the standard Davis-Putnam-Logemann-Loveland (DPLL) algorithm [10, 11] for backtrack search such as *unit propagation* and *pure literal elimination*. Note that the problem may become simple due to *different reasons* for different value assignments to the backdoor variables. Moreover, the actual semantics of the constraints may play a critical role in making the problem simple w.r.t. the sub-solver under consideration. These two aspects embedded in the notion of tractability through backdoors make this kind of structure "hidden", in contrast to other structural notions such as bounded treewidth of the underlying constraint graph. The notion of backdoors is widely applicable and has been recently investigated in the contexts of Satisfinability model counting [48], QBF-Satisfiability [55], mixed-integer programming [19], answer set programming [25], and abductive reasoning [50].

If a SAT formula F has a known small backdoor set B with respect to a tractable class C of formulas (which may, e.g., be defined as the class of formulas solvable by a polynomial-time sub-solver), then one can efficiently decide the satisfiability of F by checking the satisfiability of $2^{|B|}$ subformulas that result from all possible assignments to the variables in B and each of which belongs to the tractable class C. In other words, *branching only on the backdoor variables is essentially sufficient to solve the formula*. Directly as a solution concept, the usefulness of backdoor sets with respect to a particular tractable class depends on two factors: 1) whether problems of interest indeed have small backdoor sets with respect to the tractable class; 2) whether such small backdoor sets can be identified efficiently. We address both of these questions in this article, in the context of various underlying tractable classes and sub-solvers. We must mention, however, that the study of backdoors has also contributed *indirectly* to the design of SAT techniques—demonstrating the existence of very small backdoor sets (even if in general they cannot always be found with an efficient algorithm) has played a crucial role in motivating the use of randomization, restarts, and algorithm portfolios in existing solution approaches [59, 60]. In fact, the notion of backdoor sets came about as a way of explaining the high variance in performance of state-of-the-art SAT solvers, in particular heavy-tailed behavior [9, 31], and as a tool for analyzing and understanding the efficient performance of these solvers on many real-world instances.

Recent work has generalized the study of backdoors from algorithmic sub-solvers to any tractable class of CNF formulas. In particular, the tractable classes of 2CNF, Horn, and RHorn (renamable Horn) have received attention. Nishimura et al. [47] show that strong backdoors w.r.t. 2CNF and Horn are equivalent to so-called "deletion" backdoors with respect to these classes, where deleting all occurrences of the backdoor variables from the formula (rather than considering all possible value assignments) results in a subformula that is 2CNF or Horn, respectively. In addition, Paris et al. [49] propose a heuristic procedure for finding small deletion RHorn-backdoors (which is strictly better than considering simply Horn-backdoors). Theoretically, we show that the usefulness of deletion RHorn-backdoors is limited—they are not equivalent to strong backdoors and, in fact, there are families $\{F_n\}$ of formulas with n variables for which the smallest deletion RHorn-backdoors, as a function of n, are exponentially larger than the smallest strong RHorn-backdoors. Further, we provide integer programming encodings for finding optimal deletion Horn- and RHorn-backdoors and evaluate empirically the size of the smallest deletion backdoors for these classes. Our empirical evaluation on a set of benchmarks shows that the smallest deletion backdoors with respect to these classes are considerably larger than strong backdoors with respect to

DLL sub-solvers[1] such as unit propagation and "probing" (also known as the failed-literal rule). For example, on a set of graph coloring instances, probing results in backdoors of size less than 0.33 % of the variables while the smallest deletion/strong Horn-backdoors contain 66.67 % of the variables. These results highlight the tradeoff between the favorable complexity of finding deletion 2CNF-, Horn-, and RHorn-backdoors[2] and the large size of the smallest such backdoors in practice.

Considering tractable classes such as 2CNF, Horn and RHorn differs from backdoors with respect to DLL sub-solvers in that they do not detect trivial inconsistencies, such as having an empty clause in an *arbitrary* formula. Being able to detect the presence of an empty clause may seem like an inconsequential feature for a tractable class or a polynomial-time sub-solver underlying a backdoor set. However, we prove that adding empty-clause detection to the classes 2CNF, Horn, or RHorn increases the worst-case complexity of backdoor detection beyond the "within NP" results known for the three pure classes [7, 47] and, perhaps more importantly, can dramatically reduce the size of the resulting backdoor sets. More specifically, we prove that deciding whether a given formula has a strong 2CNF-, Horn-, or RHorn-backdoor of size k becomes both NP- and coNP-hard, and therefore strictly harder than NP assuming NP $\neq$ coNP,[3] as soon as empty-clause detection is added to these classes. In a nutshell, this change in detection complexity and in the resulting backdoor size can be attributed to the fact that empty-clause detection can be used to easily test for trivial unsatisfiability even when the remainder of the simplified formula, after assigning values to the backdoor variables, is not yet in 2CNF, Horn, or RHorn form. As an example of a significant reduction in backdoor size, we found that in certain graph coloring instances with planted cliques of size 4, while the minimum strong Horn-backdoors involve ≈ 67 % of the variables, the fraction of variables in the smallest strong backdoors w.r.t. mere empty-clause detection converges to 0 as the size of the graph grows. Theoretically, we show that there exist families of formulas for which adding empty-clause detection to the 2CNF, Horn, and RHorn classes leads to arbitrarily smaller backdoors than backdoors w.r.t. the pure 2CNF, Horn, and RHorn classes, respectively. These results again highlight the tradeoff, as a function of the underlying tractable class, between the size of the smallest backdoor sets and the computational complexity of deciding the existence of a backdoor set of a given size.

The definition of a strong backdoor set B captures the fact that a systematic tree search procedure (such as the DPLL procedure [10, 11] for SAT) restricted to branching only on variables in B will successfully solve the problem, whether satisfiable or unsatisfiable. Furthermore, in this case, the tree search procedure restricted to branching on the variables in the set B will succeed independently of the order in which it explores various parts of the search tree. In addition to using sophisticated branching heuristics, data structures, etc., most of the state-of-the-art DPLL-based SAT solvers rely heavily on *clause learning*, i.e., adding new constraints or "nogoods" every time a conflict is encountered during the tree search. Clause learning has been found to be extremely useful in practice [cf. 23, 44, 45, 51, 61], in addition enabling provably exponentially shorter proofs of unsatisfiability [3, 52].

[1]Following [57], we use the term *DLL* (or David-Logemann-Loveland) *sub-solver* to denote any polynomial-time algorithmic procedure typically employed by backtrack-style SAT solvers at every step of the search, such as unit propagation.

[2]The computational complexity of deciding the existence of a deletion backdoor of a fixed size k is only polynomial in the number of variables [47, 53].

[3]The fact that being both NP- and coNP-hard implies being harder than NP assuming NP $\neq$ coNP, is discussed in Remark 1.

Adding new information as the search progresses has, however, not been considered in the traditional concept of backdoors. In the latter part of this work, we extend the concept of backdoors to the context of learning, where information learned from previous search branches is allowed to be used by the sub-solver underlying the backdoor. This often leads to much smaller backdoors than the "traditional" ones. In particular, we prove that the smallest backdoors for SAT that take into account clause learning can be exponentially smaller than traditional backdoors oblivious to this essential solver feature. We present empirical results showing that the added power of "learning-sensitive backdoors" is also often observed in practice.

In summary, our results show that real-world SAT solvers such as *Satz* (without clause learning) and *RSat* (with clause learning) are indeed remarkably good at finding small backdoors sets. At a broader level, this work suggests that the study of structural notions that lead to efficient algorithms for combinatorial problems should consider not only "easily" identifiable *static* properties, such as having small tree-width from the outset or becoming Horn or 2CNF after assigning values to a few variables, but also *dynamic* properties that capture key aspects of state-of-the-art constraint solvers, such as unit propagation, pure literal elimination, probing, and clause learning.

Road Map The remainder of the article is organized as follows. Section 2 introduces the necessary formal definitions and provides a summary of related work (Section 2.1). Section 3 presents two of the three main theoretical results, that while relaxing key properties of strong backdoors can make identifying backdoors computationally easier (Section 3.1), it also can lead to exponentially larger backdoor sizes (Section 3.2) and hence weakens the usefulness of the notion of backdoors. Section 4 presents an empirical study of backdoor sets with respect to various notions of the underlying sub-solver. Section 5 provides theoretical results about the drastic positive effect of clause learning on backdoor size, while Section 6 supplements these results with an empirical evaluation. Finally, Section 7 concludes with a summary of the work.

2 Background and related work

In this section, we give a more formal introduction to the Boolean satisfiability problem and CNF formulas, and we introduce the notion of backdoor sets and describe several types of backdoors.

A *conjunctive normal form (CNF)* formula F is a conjunction of a finite set of clauses, a *clause* is a disjunction of a finite set of literals, and a *literal* is a Boolean variable or its negation. The literals associated with a variable x are denoted by x^ϵ, $\epsilon \in \{0, 1\}$, with x^1 denoting the "positive" literal (the one that is satisfied when $x = 1$) and x^0 denoting the "negative" literal (the one that is satisfied when $x = 0$). We sometimes specify a clause by just a set of literals, the disjunction operator being implicit. $\mathit{var}(F)$ denotes the set of variables that occur in F. A *(partial) truth assignment* (or *variable assignment*, or simply *assignment*) is a map $\tau : B \to \{0, 1\}$ defined on some set $B \subseteq \mathit{var}(F)$; if $B = \mathit{var}(F)$, τ is called a *complete* assignment. A *solution* to a CNF formula F is a complete assignment τ that satisfies all the clauses of F. For a proper subset B of $\mathit{var}(F)$, together with an assignment τ to the variables in B as well as a variable $x \in \mathit{var}(F)\setminus B$ and $\epsilon \in \{0, 1\}$, we will use $\tau \cup \{x^\epsilon\}$ to denote the assignment to the variables in $B \cup \{x\}$ that comprises the assignments τ and $x \mapsto \epsilon$. For $x \in \mathit{var}(F)$ and $\epsilon \in \{0, 1\}$, $F[\epsilon/x]$ denotes the simplified formula obtained from F by removing all clauses that contain the literal x^ϵ and removing the literal $x^{1-\epsilon}$ from the remaining clauses. For $B \subseteq var(F)$ and an assignment τ to the variables in B, $F[\tau/B]$ denotes the simplified formula obtained after setting the variables in B according to τ. Following [47], we define *deletion* of the variable x from the formula F (denoted by $F - x$) as removal of the literals of x from every clause c of F: $F - x = \left\{c \setminus \left\{x^0, x^1\right\} \mid c \in F\right\}$. For $X \subseteq var(F)$, $F - X$ is defined similarly.

A *unit clause* is a clause that contains only one literal. A *pure literal* in F is a literal x^ϵ such that $x \in var(F)$ and $x^{1-\epsilon}$ does not occur in F. A *Horn clause* is a clause that contains at most one positive literal x^1. A *binary clause* is a clause that contains exactly two literals. A formula is called *Horn* (resp., *2CNF*) if all its clauses are Horn (binary). We also use Horn and 2CNF to denote the corresponding classes of formulas. Both of these classes are tractable: Satisfiability of Horn formulas can be decided in linear time [14, 21, 32], as can satisfiability of 2CNF formulas [2, 14]. *Renaming* (or *flipping*) a variable x in F means replacing every occurrence of x^1 in F with x^0 and vice versa. F is a *renamable Horn* (RHorn) if all the clauses of F can be made Horn by flipping the variables in some subset of $var(F)$. The class of RHorn formulas is tractable: A renaming of the variables to obtain a Horn formula, if one exists, can be found in linear time [1, 14, 41].

The concept of backdoors and their theoretical foundations were introduced by Williams, Gomes, and Selman [59, 60]. Informally, a strong backdoor set for a formula F is a set $B \subseteq var(F)$ such that for every truth assignment to the variables in B, the resulting simplified formula is tractable. Williams et al. [59] formalized backdoors with respect to tractable classes of formulas, where tractability is captured in terms of a polynomial-time *sub-solver* (defined shortly). This notion of tractability is quite general, and it applies even to tractable classes for which there is no clean syntactic characterization.

Definition 1 ([59]) A set B of variables is a *strong backdoor* for a formula F w.r.t. a sub-solver S if $B \subseteq var(F)$ and for *every* truth assignment $\tau : B \to \{0, 1\}$, S returns a satisfying assignment for $F[\tau/B]$ or concludes that $F[\tau/B]$ is unsatisfiable.

Definition 2 ([59]) A set B of variables is a *weak backdoor* for a formula F w.r.t. a sub-solver S if $B \subseteq var(F)$ and for *some* truth assignment $\tau : B \to \{0, 1\}$, S returns a satisfying assignment for $F[\tau/B]$.

For brevity, we will often use the notation *S-backdoors* to refer to backdoor w.r.t. a sub-solver S. A sub-solver is a polynomial-time procedure formally defined as follows:

Definition 3 ([59]) A *sub-solver S* is an algorithm that, given a formula F as input, satisfies the following conditions:

a) *Trichotomy:* S either rejects F (gives up) or correctly determines it (as unsatisfiable or satisfiable, returning a solution if satisfiable),
b) *Efficiency:* S runs in polynomial time,
c) *Trivial solvability:* S can determine whether F is trivially true (has no clauses) or trivially false (has the empty clause, denoted by $\{\}$), and
d) *Self-reducibility:* If S determines F, then for every $x \in var(F)$ and $\epsilon \in \{0, 1\}$, S determines $F[\epsilon/x]$.

The properties of sub-solvers most relevant to our discussion will be efficiency and trivial solvability.

The notion of sub-solver introduced by Williams et al. [59] captures the key properties of SAT solvers based on the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [10, 11], a backtrack search procedure in which one systematically chooses the next variable to assign and applies polynomial-time propagation procedures (or sub-solvers) such as *unit propagation* and *pure literal elimination* to infer as many additional assignments as possible. The simplest and most *trivial sub-solver* that fulfills the conditions in Definition 3 is the one that checks for only two conditions: whether the formula is empty after simplification (and is thus trivially satisfiable), and whether it contains the empty clause (and is thus trivially unsatisfiable). We use E to denote this trivial sub-solver. More relevant sub-solvers, employed in practice by DPLL-style SAT solvers, are unit propagation (UP), pure literal elimination ($\mathrm{PL}^{\{\}}$), and their combination (UP+PL). Yet another, more advanced, propagation technique based on lookahead is probing (PROB). We formally describe these sub-solvers in Table 1.

Every sub-solver S correctly determines a particular sub-class of CNF formulas and rejects others, which implicitly defines the class of formulas that it can determine in polynomial time. A natural variant of the definition of a backdoor does not explicitly appeal to a sub-solver, but rather requires the formula that remains after values are assigned to the backdoor variables to fall within a known tractable class of CNF formulas such as 2CNF, Horn, or RHorn. We will refer to such backdoors as 2CNF-backdoors, Horn-backdoors, and RHorn-backdoors, respectively.[4]

In general, for any class C of tractable formulas, one may thus speak of a*strong C-backdoor*, i.e., a set of variables such that for every truth assignment to them, the resulting simplified formula belongs to the class C. Note that this way of defining backdoors *de facto* corresponds to relaxing the assumption of trivial solvability for a sub-solver . The tractable class implicitly defined by any given sub-solver includes by definition all formulas that are trivially satisfiable or trivially unsatisfiable, while many well-known tractable classes C do not include such formulas. For example, an arbitrary formula containing the empty clause may not be Horn. However, such formulas—with the empty clause in them—are important for our discussion. We use E to denote not only the trivial sub-solver but also the class of formulas that it decides, that is, the empty formula and all formulas that contain the empty clause.

Definition 4 E is the class of all CNF formulas that contain the empty clause, {}. For any class C of formulas, $C^{\{\}}$ denotes the class $C \cup E$.

A *$C^{\{\}}$-backdoor* thus corresponds to a set of variables such that for every assignment to the variables in the backdoor, the resulting simplified subformula either contains the empty clause or belongs to C. Table 1 summarizes all the tractable CNF classes that are relevant to the study of backdoors in this work.

[4]One may, of course, consider other tractable sub-classes as well. However, certain sub-classes such as that of CNF translations of XOR constraints (i.e., linear equations modulo 2 represented as a set of clauses) are not very suitable as a base class for defining backdoors. This is because, even though formulas representing a set of XOR constraints can be solved efficiently using Gaussian elimination, *identifying* whether the residual formula (after assigning truth values to the backdoor variables) falls in this sub-class is itself computationally challenging, limiting the practical utility of XOR-backdoors in a SAT solver.

Table 1 Sub-solvers and tractable classes w.r.t. which backdoors are studied in this work. Top section: propagation based or DLL sub-solvers. Middle section: syntactic tractable classes. Bottom section: syntactic classes with empty-clause detection

E	The sub-solver simply checks for the empty formula and the empty clause and determines the formula accordingly. If neither holds, the formula is rejected.
$\mathrm{PL}^{\{\}}$	The pure literal elimination sub-solver checks for variables that appear as pure literals, and then assigns each of them the corresponding value and simplifies, until the formula is trivially solvable (notice that PL cannot decide unsatisfiable instances). If the remaining formula is not trivially solvable but contains no more pure literals, then it is rejected. To fulfill the definition of sub-solver, every formula is also first checked for the empty clause. For simplicity, we will refer to this sub-solver as PL.
UP	Given a formula F, the unit propagation sub-solver (UP) checks whether the formula is empty or contains the empty clause, in which case it is trivially solvable; otherwise, it checks whether the formula contains a unit clause. If the formula contains no unit clauses, it is rejected; otherwise, it assigns the variable in the unit clause the corresponding satisfying value and recurses on the simplified formula.
UP+PL	The sub-solver first applies unit propagation until there are no more unit clauses and then applies the pure literal rule. If the formula is not simplified to one that is trivially solvable or trivially inconsistent, it is rejected.
PROB	Probing, also known as the *failed literal rule*, first applies UP. If the formula is not determined, then true and false are assigned separately to some unassigned variable and UP is applied to the resulting subformulas. If either of the two assignments results in a complete satisfying assignment, the formula is deemed satisfiable. If both assignments result in an unsatisfiable subformula, then the formula is determined unsatisfiable. If only one of the two assignments results in an unsatisfiable subformula, then the other value is assigned to that variable and the procedure recurses. Otherwise, another unassigned variable is probed until all unassigned variables are probed. If the formula is not simplified to one that is trivially solvable or trivially inconsistent, it is rejected.[a]
PROB +PL	This sub-solver applies probing as above, but at each probing assignment it uses UP+PL as the subprocedure instead of UP.
2CNF	First checks for membership by scanning all clauses and checking whether each contains two literals. If this check fails, the formula is rejected; otherwise, the formula is decided in linear time [e.g., 2, 14].
Horn	First checks for membership by scanning all clauses and checking whether each contains at most one positive literal. If this check fails, the formula is rejected; otherwise, the formula is decided by running unit propagation and setting any remaining variables to 0 [32] [see also 14].
RHorn	First checks for membership by finding a valid variable renaming if one exists; this can be accomplished with known linear-time algorithms [1, 14, 41]. If this check fails, the formula is rejected; otherwise, the resulting Horn formula is solved.
$\mathrm{2CNF}^{\{\}}$	First checks for empty clause. If that does not apply, then checks 2CNF.
$\mathrm{Horn}^{\{\}}$	First checks for empty clause. If that does not apply, then checks Horn.
$\mathrm{RHorn}^{\{\}}$	First checks for empty clause. If that does not apply, then checks RHorn.

[a]This sub-solver will need to apply UP at most $O(n^2)$ times. Each time a variable is set by probing, we must re-probe all the remaining unassigned variables

2.1 Related work

Several studies have demonstrated an empirical correlation between the size of the smallest strong backdoors and the search effort required by SAT solvers. Lynce and Marques-Silva [43] show that the search effort required by the SAT solver *zChaff* [45, 61] to prove a random 3-SAT formula unsatisfiable is correlated with the size of the strong backdoors w.r.t. the trivial sub-solver E. Kilby et al. [39] study strong SATZ-backdoors: sets B of variables such that for every assignment τ to these variables, the SAT solver *Satz* [42] solves the simplified formula $F[\tau/B]$ without making any branching decisions (i.e., with a "branch-free" search). *Satz* is a DPLL-based SAT solver that incorporates a strong variable selection heuristic and a propagation sub-solver that applies a limited form of PROB+PL (probing is not applied recursively—the variables are scanned only once, even if probing yields a successful assignment). Kilby et al. [39] measure problem hardness, defined as the logarithm of the number of search nodes required by *Satz*, and find that it is correlated with the size of the smallest strong SATZ-backdoors.

Computing a *smallest backdoor* is in general intractable in the worst case. This intractability result assumes that the size of the smallest backdoors is unknown and can be as large as n, the number of variables. However, one can ask the decision question of whether a given formula F has a backdoor of size at most k. We refer to this decision problem as the problem of STRONG S-BACKDOOR DETECTION for sub-solvers (and, similarly, STRONG C-BACKDOOR DETECTION for tractable classes).

STRONG S-BACKDOOR DETECTION
Input: A formula F
Parameter: A positive integer k
Question: Does F have a strong backdoor of size at most k with respect to sub-solver S?

For any sub-solver S, given $\langle F, k\rangle$ as input, the problem of deciding whether F has a strong S-backdoor of size k is in the complexity class Σ_2^{P} as we can (informally) formulate it as: Does there *exist* a backdoor set $B \subseteq var(F), |B| = k$, such that for *every* truth assignment $\tau : B \to \{0,1\}$, S correctly determines $F[\tau/B]$?[5] If the size k of the backdoor is small compared to the number n of variables in F, one can search for the backdoor in a brute-force manner by considering all $\binom{n}{k}$ subsets of k variables and all 2^k truth assignments to these candidate variables. This is technically a polynomial-time process for a fixed value of k, although even for moderate values of k the runtime becomes infeasible in practice. Can one do better? This is a question considered in the area of parameterized complexity theory [22].

An instance of a parameterized problem is a pair $\langle I, k\rangle$ where I is the main part and k is the parameter. A parameterized problem is called *fixed-parameter tractable* (FPT) w.r.t. k

[5]Here is how one may formally define the problem of STRONG (or WEAK) S-BACKDOOR DETECTION. Let the variables of F be $X = \{X_1, X_2, ..., X_n\}$. For $I \subseteq \{1, 2, ..., n\}$, let X_I denote $\{X_i \in X \mid i \in I\}$, that is, the subset of variables indexed by the indices in I. Let $g_S(F)$ be a predicate that evaluates to True if S correctly determines F, and to False otherwise. Since S is a polynomial-time sub-solver, $g_S(F)$ can be evaluated in (deterministic) polynomial time. With this notation, we can formalize WEAK S-BACKDOOR DETECTION as the NP problem of determining the truth value of the following formula: $\exists I \subseteq \{1, 2, ..., n\} : \big(|I| = k \text{ and } \exists V \in \{0,1\}^k : g_S(F[V/X_I])\big)$. Similarly, we can formalize STRONG S-BACKDOOR DETECTION as the Σ_2^P problem of determining the truth value of the following formula: $\exists I \subseteq \{1, 2, ..., n\} : \big(|I| = k \text{ and } \forall V \in \{0,1\}^k : g_S(F[V/X_I])\big)$.

if it can be solved in time $O(f(k)||I||^c)$ where f is any computable function, $||I||$ denotes the size of I, and c is a constant (independent of k). Parameterized complexity[6] offers a completeness theory, similar to the theory of NP-completeness, described by a hierarchy of parameterized-complexity classes, which are defined as the closure of certain parameterized problems under parameterized reductions:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq XP.$$

A parameterized problem known to be W[t]-hard (under parameterized reductions) for some $t \geq 1$ is very unlikely to be fixed-parameter tractable (i.e., the class containment is probably strict). Fixed-parameter tractability of a W[t]-hard problem would imply that the Exponential Time Hypothesis fails [26] (i.e., it would imply the existence of a $2^{o(n)}$ algorithm for n-variable 3SAT). A canonical example of a fixed-parameter tractable problem is Vertex Cover, while Clique is W[1]-complete, and Set Cover is W[2]-complete, with the parameter k as the size of the solution [22].

The parameterized version of STRONG S-BACKDOOR DETECTION treats the size k of the backdoor as a parameter. Note that if backdoor detection for some tractable class or sub-solver is shown to be fixed-parameter tractable, then one can decide the existence of a backdoor set of size k in time $O(f(k)n^c)$, where c does not depend on k—as opposed to the runtime of the brute-force approach discussed earlier—meaning that in principle one could search fairly efficiently for potentially large backdoors.

Szeider [57] studied the complexity of finding strong backdoors w.r.t. DLL sub-solvers. For $S \in \{$UP, PL, UP+PL$\}$ and with k as the parameter of interest, he proved that the problem of deciding whether there exists a strong S-backdoor of size k is complete for the parameterized complexity class W[P].[7] Interestingly, the naïve brute-force procedure for this problem already has complexity $O(n^k 2^k n^\alpha)$ and works by enumerating all subsets of size $\leq k$, trying all assignments for each such subset, and running the $O(n^\alpha)$ time sub-solver. This, combined with the W[P]-completeness of the problem, indicates that asymptotically, in terms of worst-case behavior, we are unlikely to find a smallest strong backdoor w.r.t. UP, PL, or UP+PL more efficiently than with brute-force search [57].

Nishimura et al. [47] studied the complexity of finding strong backdoors with respect to the classes 2CNF and Horn. They observed that an important general property of these two classes is the following:

Definition 5 A class C of formulas is *closed under removal of clauses* if removing arbitrary clauses from any formula in C keeps the formula in C.

For tractable classes of formulas that are closed under clause removal, one can consider a different notion of backdoors, one that involves a set of variables such that once these variables are "deleted" from the formula, the remaining formula falls into a given tractable class (without considering any simplification due to truth assignments).

Definition 6 ([47]) A set B of variables is a *deletion backdoor* for a formula F w.r.t. a class C if $B \subseteq var(F)$ and $F - B \in C$.

[6]For a detailed treatment of the theory of parameterized complexity, we refer the reader to [22].

[7]A discussion of parameterized complexity classes such as W[P] is beyond the scope of this article, except for the mention that W[P] contains FPT and the containment is generally believed to be strict.

Deletion backdoors for classes satisfying closure under clause deletion are useful in capturing structure in a backtrack search solver because of the following observation, which implies that restricting branching to the variables of a deletion backdoor (w.r.t. tractable classes closed under clause deletion) is sufficient to efficiently solve the formula.

Proposition 1 ([47]) *Let C be a tractable class of CNF formulas that is closed under clause removal. Then, every deletion C-backdoor is also a strong C-backdoor. In particular, this holds for $C \in \{Horn, 2CNF\}$.*

In general, strong C-backdoors are not deletion C-backdoors, because assigning values to variables (as opposed to deleting variables) usually leads to further simplification of the formula. We will show, for example, that there is an exponential separation between the size of the smallest deletion RHorn-backdoors and the smallest strong RHorn-backdoors. Nonetheless, Nishimura et al. [47] show that for $C \in$ {2CNF, Horn}, deletion backdoors and strong backdoors are equivalent.

In terms of the problem of identifying a deletion backdoor of a certain size, the following proposition formalizes the easy to observe fact that given a candidate deletion backdoor, verifying its validity is a straightforward task. This, as we will later see, is not always possible for a strong backdoor (unless P=NP).

Proposition 2 *When membership in C can be checked in polynomial time, the problem of deciding whether F has a deletion C-backdoor of size k is in NP.*

For $C \in$ {2CNF, Horn}, Nishimura et al. [47] show that the problem of deciding whether F has a deletion C-backdoor of size k is, in fact, NP-complete but also fixed parameter tractable (FPT)—in particular, that it can be solved in time $O(2^k n)$ and $O(3^k n)$ for Horn and 2CNF respectively, where n is the number of literals that occur in F. Together with the result that strong Horn-backdoors are also deletion Horn-backdoors, and that strong 2CNF-backdoors are also deletion 2CNF-backdoors, this shows that strong {Horn,2CNF} -backdoor detection is FPT, in contrast to the W[P]-completeness result for strong {UP, PL, UP+PL}-backdoor detection mentioned earlier.

Another tractable class of CNF formulas that is closed under clause removal is RHorn. Hence by Proposition 1, deletion RHorn-backdoors are also strong RHorn-backdoors. Similarly to Horn, the problem of deciding whether F has a deletion RHorn-backdoor of size k is NP-complete [7] and has recently been shown to also be FPT, using a result for MAX-2-SAT [53]. Two studies have proposed heuristic methods for finding small deletion RHorn-backdoors. Paris et al. [49] proposed a two-step heuristic approach for finding deletion RHorn-backdoors that are not necessarily of minimal size. They found that branching on the variables occurring in the backdoors detected with their approach can significantly speed up DPLL solvers. Later, Kottler et al. [40] introduced two additional ways of computing deletion RHorn-backdoors, both of which generally outperform the method of [49]. The authors provide empirical results regarding backdoors sizes for several SAT benchmark instances.

3 Theoretical results for various sub-subsolvers

While other approaches to studying structural properties of combinatorial problems have looked at instance characteristics that are "statically" identifiable, such as the topology of the underlying constraint graph, the notion of backdoors captures "hidden" structure. A key property of the strong backdoor definition is that it allows for "dynamic" constraint reasoning as a function of variable assignments: for *every assignment* to the backdoor variables, the resulting simplified formula should be decidable by the sub-solver. "Static" properties, on the other hand, can be evaluated without considering variable assignments. Examples of such properties are verifying whether a variable subset $B \subseteq X$ is a cutset, or whether deleting the variables in B from the formula results in a Horn/RHorn/2CNF sub-formula.

Another important property of backdoors is that the definition of a sub-solver captures inconsistency detection (the solvability property), a key feature underlying the efficiency of SAT solvers. In this section, we show that while relaxing each of these properties can make identifying backdoors computationally easier (Section 3.1), it also can lead to exponentially larger backdoor sizes (Section 3.2) and hence weakens the usefulness of the notion of backdoors.

3.1 The impact of empty-clause detection

First, we show that strong backdoors w.r.t. $2CNF^{\{\}}$, $Horn^{\{\}}$, and $RHorn^{\{\}}$ behave very differently, in terms of both the complexity of finding backdoors and the backdoor size, compared to strong backdoors w.r.t. 2CNF, Horn, and RHorn, respectively.[8] In particular, we prove that the problem of deciding whether a formula has a strong backdoor of size k w.r.t. $C \in \{2CNF^{\{\}}, Horn^{\{\}}, RHorn^{\{\}}\}$ is NP-hard and coNP-hard. This shows that unless NP = coNP, this problem is much harder[9] than determining whether there exist strong backdoors of size k w.r.t. $C \in$ {2CNF, Horn, RHorn}, which is known to be NP-complete and is thus within NP [7, 47].

Before we present our worst-case complexity results, we start with a relatively simple observation, which highlights the potential positive impact of considering empty-clause detection as in the classes $2CNF^{\{\}}$, $Horn^{\{\}}$, and $RHorn^{\{\}}$. Consider the following unsatisfiable family $\{F_n\}_{n\geq 4}$ of formulas:

$$F_n = (x_1) \wedge (\neg x_1) \wedge (x_1 \vee x_2 \vee \ldots \vee x_n) \wedge (\neg x_1 \vee \neg x_2 \vee \ldots \vee \neg x_n)$$

It is clear that the variable x_1 is the "cause" of F_n being unsatisfiable. Indeed, $\{x_1\}$ is a strong E-backdoor of size 1 for F_n, and hence also a strong $C^{\{\}}$-backdoor for any class C. On the other hand, one must assign values to at least $n - 1$ variables before the third clause of F_n becomes Horn. Similarly, one must assign values to at least $n - 2$ variables before the third and fourth clauses of F_n become 2CNF or RHorn.[10] This example shows that empty-clause detection can lead to not only *exponentially* smaller but even *arbitrarily* smaller backdoor sets than those with respect to the classes 2CNF, Horn, and RHorn.

Proposition 3 *Let C be any class of formulas. There are families $\{F_n\}_{n\geq 4}$ of formulas with a strong $C^{\{\}}$-backdoor of size 1 but for which all strong Horn-, RHorn-, or 2CNF-backdoors are of size at least $n - 2$.*

[8] Recall that adding the set E of all formulas with an empty clause to 2CNF, Horn, or RHorn corresponds to adding empty-clause detection to the sub-solver for that class.

[9] This is further discussed in Remark 1.

[10] For RHorn, if we assign values to at most $n - 3$ variables of F_n, then under every renaming, at least two of the remaining variables will be positive in at least one of the two long clauses of F_n, violating the Horn property.

Let us return to the subject of analyzing the worst-case complexity of finding small backdoors when empty-clause detection is added to the underlying class. To prove the NP-hardness result mentioned above, we extend the argument originally used by Nishimura et al. [47] for (pure) 2CNF- and Horn-backdoors, employing a reduction from the NP-complete Vertex Cover problem. In the other direction, we prove coNP-hardness of backdoor detection w.r.t. these classes, enriched by empty-clause detection, by directly reducing UNSAT, the coNP-complete problem of deciding whether a given CNF formula is unsatisfiable, to strong $C^{\{\}}$-backdoor detection, exploiting the fact that the classes Horn, RHorn, and 2CNF are closed under clause deletion. These two results are formally stated as Lemmas 1 and 2 below, which together lead to our first main result, Theorem 1.

Lemma 1 *Let $C \in \{$Horn, RHorn, 2CNF$\}$. Given a formula F and $k \geq 0$, the problem of deciding whether F has a strong $C^{\{\}}$-backdoor of size k is NP-hard.*

Proof We extend the argument originally used by Nishimura et al. [47] for (pure) 2CNF- and Horn-backdoors. The polynomial-time reduction is from the NP-complete Vertex Cover problem: Given an undirected graph $G = (V, E)$ and a number $k \geq 0$, does G have a vertex cover of size k? A vertex cover U is a subset of V such that every edge in E has at least one endpoint in U. Given an instance $\langle G, k\rangle$ of this problem, we will construct a formula F_{Horn} with $n \geq k$ variables and with all positive literals such that F_{Horn} has a strong Horn$^{\{\}}$-backdoor of size k if and only if G has a vertex cover of size k. Similarly, we will construct F_{RHorn} and F_{2CNF}.

(I) F_{Horn} has $|V|$ variables and $|E|$ clauses. The variables are x_v for each $v \in V$ and the clauses are defined as follows:

$$F_{\text{Horn}} = \bigwedge_{\substack{\{u,v\}\in E\\ u<v}} (x_u \vee x_v)$$

It is easy to see that if G has a vertex cover U of size k, then the corresponding variable set $X_U = \{x_u \mid u \in U\}$ is a strong Horn-backdoor, and hence a strong Horn$^{\{\}}$-backdoor, of size k: for any assignment τ to X_U, $F_{\text{Horn}}[\tau/X_U]$ contains clauses of length at most one, and is thus trivially in Horn (and thus in Horn$^{\{\}}$). For the other direction, suppose X_U is a strong Horn$^{\{\}}$-backdoor of size k. We claim that the variables in X_U must touch every clause of F_{Horn}, implying that the corresponding vertices U touch every edge of G and thus form a vertex cover of size k. To see this, consider the all-1's assignment τ_1 to X_U. Since X_U is a strong Horn$^{\{\}}$-backdoor and assigning variables according to τ_1 cannot result in creating the empty clause, $F_{\text{Horn}}[\tau_1/X_U]$ must be Horn. If X_U did not touch a clause c of F_{Horn}, then c would appear in $F_{\text{Horn}}[\tau_1/X_U]$ as a binary clause with two positive literals, violating the Horn property. Hence, the claim holds.

(II) F_{RHorn} has $|V| + |E|$ variables and $4|E|$ clauses. The variables are x_v for each $v \in V$ and $y_{u,v}$ for each $\{u, v\} \in E$, $u < v$. The clauses of F_{RHorn} are defined as follows:

$$F_{\text{RHorn}} = \bigwedge_{\substack{\{u,v\}\in E\\ u<v}} \begin{pmatrix} (x_u \vee x_v \vee \neg y_{u,v}) \wedge (x_u \vee \neg x_v \vee \neg y_{u,v}) \wedge \\ (\neg x_u \vee x_v \vee \neg y_{u,v}) \wedge (\neg x_u \vee \neg x_v \vee \neg y_{u,v}) \end{pmatrix}$$

Note that the only clauses of F_{RHorn} violating the Horn property are the $|E|$ clauses of the form $(x_u \vee x_v \vee \neg y_{u,v})$. Note also that F_{RHorn} is not a renamable Horn formula, as any renaming of the x variables will leave at least one clause per edge in non-Horn form. As before, if G has a vertex cover U of size k, then the corresponding variable set $X_U = \{x_u \mid u \in U\}$ is in fact a strong Horn-backdoor, and hence also a strong RHorn$^{\{\}}$-backdoor of size k: For any assignment τ to X_U, the $|E|$ clauses of F_{RHorn} violating the Horn property either become satisfied or are reduced to having at most one positive literal, and thus become Horn. For the other direction, suppose $B = X_U \cup Y$ is a strong RHorn$^{\{\}}$-backdoor for F_{RHorn} of size k, where Y denotes a (possibly empty) subset of the y variables. We claim that variables of X_U must touch every clause of F_{RHorn}, implying that the corresponding vertices U touch every edge of G and thus form a vertex cover of size $\leq k$. (Since $k \leq n$, we can extend U to a vertex cover of size exactly k by adding to it $k - |U|$ arbitrary vertices of G.) To see this, suppose for the sake of contradiction that the vertex set U does not touch edge (u, v) of G and is therefore not a valid vertex cover. In other words, B contains neither x_u nor x_v. Consider the truth assignment τ that sets $B \setminus \{y_{u,v}\}$ to 0 and $B \cap \{y_{u,v}\}$ to 1. In words, this says that τ assigns all variables of B the value 0, except for $y_{u,v}$, which is assigned the value 1 if it is part of B (and left unset otherwise). All $4(|E| - 1)$ clauses of F_{RHorn} not involving $y_{u,v}$ contain at least one negative literal and therefore cannot generate an empty clause in $F_{\text{RHorn}}[\tau/B]$. By assumption, B contains neither x_u nor x_v, and thus the 4 clauses involving $y_{u,v}$ also do not generate the empty clause in $F_{\text{RHorn}}[\tau/B]$. Since B is a strong RHorn$^{\{\}}$-backdoor, it follows that $F_{\text{RHorn}}[\tau/B]$ must be in RHorn. However, this formula contains either the 4 clauses

$$(x_u \vee x_v), (x_u \vee \neg x_v), (\neg x_u \vee x_v), (\neg x_u \vee \neg x_v)$$

or the 4 clauses,

$$(x_u \vee x_v \vee \neg y_{u,v}), (x_u \vee \neg x_v \vee \neg y_{u,v}), (\neg x_u \vee x_v \vee \neg y_{u,v}), (\neg x_u \vee \neg x_v \vee \neg y_{u,v}),$$

depending on whether or not $y_{u,v} \in B$, and is therefore not in RHorn in either case because under any renaming at least one of the 4 clauses will have at least two positive literals—a contradiction. It follows that B must have included at least one of x_u and x_v to begin with, and thus U is a valid vertex cover for G, as desired.

(III) $F_{2\text{CNF}}$ has $|V| + |E|$ variables and $|E|$ clauses. The variables are x_v for each $v \in V$ and $y_{u,v}$ for each $\{u, v\} \in E, u < v$. The clauses of $F_{2\text{CNF}}$ are defined as follows:

$$F_{2\text{CNF}} = \bigwedge_{\substack{\{u,v\}\in E \\ u<v}} (x_u \vee x_v \vee y_{u,v})$$

The argument for the correctness of the reduction is very similar to the one for F_{Horn}, relying on the all-1's assignment. The only difference is that if we have a strong backdoor X_U, it may contain some of the y variables, and so there is no direct way of obtaining a vertex cover out of X_U. However, this is easy to fix. If X_U contains $y_{u,v}$ and also at least one of x_u and x_v, we can simply disregard $y_{u,v}$ when constructing a vertex cover. If X_U contains $y_{u,v}$ but neither x_u nor x_v, we can replace $y_{u,v}$ with either of these two variables and obtain a backdoor set of the same size but with fewer (and eventually no) such y variables. □

We now prove coNP-hardness of backdoor detection w.r.t. Horn$^{\{\}}$, RHorn$^{\{\}}$, and 2CNF$^{\{\}}$. In fact, the result holds for any class C having two natural properties. The first property, "closure under clause removal," was introduced in Definition 5, and the other property is as follows:

Definition 7 A class C of formulas is said to *support large strong backdoors* if there exists a polynomial (in k) time constructible family $\{G_k\}_{k\geq 0}$ of formulas such that the smallest strong C-backdoors for G_k have size larger than k.

In particular, 2CNF$^{\{\}}$ and Horn$^{\{\}}$ support large strong backdoors as witnessed by the following simple single-clause family of formulas: $G_k = (x_1 \vee x_2 \vee \ldots \vee x_{k+3})$. It can be easily verified that the all-0's assignment to any subset of k variables of G_k leaves a clause with three positive literals, which is neither 2CNF nor Horn nor an empty clause. Hence, the smallest 2CNF$^{\{\}}$- or Horn$^{\{\}}$-backdoors for G_k have size larger than k.

RHorn$^{\{\}}$ also supports large strong backdoors, although the argument is slightly more involved. Consider the following family of formulas over $3(k+1)$ variables and $2(k+1)$ clauses:

$$G_k = \bigwedge_{1 \le i \le k+1} ((x_i \vee y_i \vee z_i) \wedge (\neg x_i \vee \neg y_i \vee \neg z_i))$$

Suppose G_k has an RHorn$^{\{\}}$-backdoor B of size k. Clearly, there is at least one $i \in \{1, 2, \ldots, k+1\}$ such that none of x_i, y_i, and z_i belongs to B. Then, for any assignment τ to B, and in any renaming of x_i, y_i, and z_i, at least one of the two clauses involving them will have at least two positive literals, and will thus not be Horn violating the RHorn property of $G_k[\tau/B]$. Then, since B is a strong RHorn$^{\{\}}$-backdoor, for every assignment τ to B, $G_k[\tau/B]$ must contain an empty clause. For any k variables that participate in B, it can be easily verified that an assignment τ to B, where $\tau(x_i) = 0$ for all $x_i \in B$ and $\tau(y_i) = 1$ for all $y_i \in B$, does not lead to an empty clause in $G_k[\tau/B]$. Thus, the smallest RHorn$^{\{\}}$-backdoors for G_k are of size larger than k, as desired.

Lemma 2 *Let C be a class of formulas such that (1) C is closed under removal of clauses and (2) $C^{\{\}}$ supports large strong backdoors. Then, given a formula F and $k \geq 0$, the problem of deciding whether F has a strong $C^{\{\}}$-backdoor of size k is coNP-hard.*

Proof Let UNSAT denote the coNP-complete problem of deciding whether a given CNF formula is unsatisfiable. We prove the lemma by reducing UNSAT to $C^{\{\}}$-backdoor detection. Let H be a CNF formula over variables V_H, $|V_H| = k$. We create a formula F such that F has a strong $C^{\{\}}$-backdoor of size k if and only if H is unsatisfiable. The idea is to start with H and append to it a formula on a disjoint set of variables such that for any assignment to k backdoor variables, the combined formula does not reduce to a formula in C and must therefore contain the empty clause in order to belong to $C^{\{\}}$.

F is constructed as follows. Using the fact that $C^{\{\}}$ supports large strong backdoors, construct in polynomial time a formula G over a disjoint set of variables (i.e., variables not appearing in H) such that G does not have a strong $C^{\{\}}$-backdoor of size k. Now let $F = H \wedge G$. We prove that $H \in$ UNSAT if and only if F has a strong $C^{\{\}}$-backdoor of size k.

($\Rightarrow$) Suppose H is unsatisfiable. This implies that every truth assignment τ to V_H, the variables of H, violates some clause of H. It follows that for each such τ, $F[\tau/V_H] = H[\tau/V_H] \wedge G[\tau/V_H]$ contains the empty clause and is therefore in $C^{\{\}}$. Hence V_H gives us the desired backdoor of size k.

($\Leftarrow$) Suppose F has a strong $C^{\{\}}$-backdoor B of size k. Partition B into $B_H \cup B_G$, where B_H has the variables of H and B_G has the variables of G. By the construction of G and because $|B_G| \leq k$, B_G cannot be a strong $C^{\{\}}$-backdoor for G. In other words, there exists an assignment τ_G to B_G such that $G[\tau_G/B_G] \notin C$ and $G[\tau_G/B_G]$ does not contain an empty clause. Because of the closure of C under removal of clauses and the variable disjointness of H and G, it follows that $F[\tau/B] \notin C$ for every extension $\tau = (\tau_H, \tau_G)$ of τ_G to all of B. However, since B is a strong $C^{\{\}}$-backdoor for F, it must be that $F[\tau/B] \in C^{\{\}}$, and the only possibility left is that $F[\tau/B] \in$ E. Since $G[\tau_G/B_G] \notin$ E, it must be that $H[\tau_H/B_H] \in E$ for all such extensions τ of τ_G. In words, this says that $H[\tau_H/B_H]$ contains a violated clause for every truth assignment to B_H. Therefore, H is unsatisfiable as desired. □

Lemmas 1 and 2 together give us our first main theorem:

Theorem 1 *Let $C \in \{$Horn, RHorn, 2CNF$\}$. Given a formula F and $k \geq 0$, the problem of deciding whether F has a strong $C^{\{\}}$-backdoor of size k is both NP-hard and coNP-hard.*

Remark 1 It is well known that if a problem $\mathscr{P}$ is both NP-hard and coNP-hard, then it is harder than both NP and coNP assuming NP $\neq$ coNP. For completeness, we briefly review the argument here. Suppose for the sake of contradiction that $\mathscr{P}$ is in NP while the above assumptions hold. Since $\mathscr{P}$ is assumed to be coNP-hard, this implies that all problems in coNP are also in NP, that is, coNP $\subseteq$ NP. We will show that the reverse must also hold, i.e., NP $\subseteq$ coNP, together implying NP $=$ coNP—a contradiction to one of our starting assumptions. Since $\mathscr{P}$ is assumed to be in NP, its complement, $\overline{\mathscr{P}}$, is, by definition, in coNP. Since we had coNP $\subseteq$ NP, we also have that $\overline{\mathscr{P}}$ is in NP. This, by definition, implies that the complement of $\overline{\mathscr{P}}$, which is $\mathscr{P}$ itself, is in coNP. Similar to the earlier reasoning, $\mathscr{P}$ being NP-hard and in coNP implies that all problems in NP are also in coNP, that is, NP $\subseteq$ coNP, as desired. This finishes the proof that if $\mathscr{P}$ is both NP-hard and coNP-hard, and is in NP, then it must be that NP $=$ coNP. A symmetric argument works if $\mathscr{P}$ is assumed to be in coNP.

3.2 Strong backdoors vs. deletion backdoors

Let us turn our attention to the relationship between strong and deletion backdoors. While these two kinds of backdoors are known to be equivalent for the classes 2CNF and Horn, we prove an exponential separation for a slightly stronger class that has a slight "dynamic" flavor to it—that of renamable-Horn or RHorn formulas, where the specific renaming used to convert a sub-formula to the Horn form may differ depending on the variable restriction that yields that sub-formula. Specifically, we demonstrate the existence of formulas for which strong RHorn-backdoors are exponentially smaller than the smallest deletion RHorn-backdoors. This suggests that strong RHorn-backdoors are more likely to succinctly capture structural properties of interest in formulas than strong backdoors w.r.t. purely static classes like Horn.

The main idea behind the proof is the following. Suppose B is a strong RHorn-backdoor for F. Then for each assignment τ to the variables in B, there exists a renaming r_τ such that $F[\tau/B]$ under the renaming r_τ yields a Horn formula. If F is such that for different τ, the various renamings r_τ are different and mutually incompatible, then there is no single renaming r under which $F - B$, the formula obtained by deleting the variables in B, becomes Horn. The following example illustrates this point, which we will generalize to an exponential separation in the proof of Theorem 2.

Example 1 Consider the following formula:

$$\begin{aligned} F \;=\; & (x_1 \vee x_2) \wedge (\neg y_1 \vee \neg y_2) \wedge (\neg x_1 \vee y_1 \vee z) \wedge (\neg x_1 \vee y_2 \vee \neg z) \\ & \wedge (\neg x_2 \vee y_1 \vee z) \wedge (\neg x_2 \vee y_2 \vee \neg z) \end{aligned}$$

First, observe that $B = \{z\}$ is a strong RHorn-backdoor for F. This is because for $z = 0$ we can rename x_1 and y_1, and for $z = 1$ we can rename x_1 and y_2, and obtain a Horn formula in each case. On the other hand, $\{z\}$ certainly does not work as a deletion RHorn-backdoor because we must rename at least one of x_1 and x_2, which, after deleting z from all clauses that it appears in, forces both y_1 and y_2 to be renamed, resulting in a renaming under which the second clause of F does not satisfy the Horn property. In fact, it can be easily verified that both $\{x_1\}$ and $\{y_1\}$ are also not valid deletion RHorn-backdoors. From the symmetry between x_1 and x_2 and between y_1 and y_2, it follows that F does not have a deletion RHorn-backdoor of size 1.

Theorem 2 *There are formulas for which the smallest strong RHorn-backdoors are exponentially smaller than any deletion RHorn-backdoors.*

Proof Let s be a power of 2, $t = s + \log_2 s$, and $n = s + \log_2 s + t = 2 \cdot (s + \log_2 s)$. We will prove the theorem by explicitly constructing a family of formulas $\{F_n\}$ such that F_n is defined over n variables, F_n has a strong RHorn-backdoor of size $\log_2 s = \Theta(\log n)$, and every deletion RHorn-backdoor for F_n is of size at least $s + \log_2 s - 1 = \Theta(n)$.

F_n is constructed on three kinds of variables: $\{x_i \mid 1 \leq i \leq t\}$, $\left\{y_j \mid 1 \leq j \leq s\right\}$, and $\left\{z_k \mid 1 \leq k \leq \log_2 s\right\}$. Variables z_k are used to encode all s 0-1 sequences of length $\log_2 s$. Specifically, for $1 \leq j \leq s$, let D_z^j be the unique clause involving all z variables where each z_k appears negated in D_z^j if and only if the k^{th} bit of j, written in the binary representation, is a 1. For example, for $j = 01101$, $D_z^j = (z_1 \vee \neg z_2 \vee \neg z_3 \vee z_4 \vee \neg z_5)$. Note that D_z^j is falsified precisely by the unique assignment that corresponds to the binary representation of j. F_n is defined to have exactly the following $st + 2$ clauses:

$$\begin{aligned}
C_x &\equiv (x_1 \vee x_2 \vee \ldots \vee x_t) \\
C_y &\equiv (\neg y_1 \vee \neg y_2 \vee \ldots \vee \neg y_s) \\
C_z^{i,j} &\equiv \left(\neg x_i \vee y_j \vee D_z^j\right) \qquad\qquad \forall i \in \{1, \ldots, t\}, j \in \{1, \ldots, s\}
\end{aligned}$$

We now analyze RHorn-backdoors for F_n. First, we show that $\left\{z_k \mid 1 \leq k \leq \log_2 s\right\}$ is a strong RHorn-backdoor for F_n. To see this, fix any assignment $\tau \in \{0, 1\}^{\log_2 s}$ to the z variables. By the discussion above, τ satisfies all but one clause D_z^j. Let us denote this falsified clause by D_z^τ. It follows that the reduced formula, $F_n[\tau/z]$, consists of C_x, C_y, and for each $i \in \{1, \ldots, t\}$, the binary clause $(\neg x_i \vee y_\tau)$. We can convert this formula to Horn by renaming or flipping the signs of all x_i, and of y_τ. This renaming makes C_x Horn. Further, it preserves the Horn property of C_y as well as of each of the t residual binary clauses. Hence the z variables form a strong RHorn-backdoor of size $\log_2 s$.

To derive a lower bound on the size of every deletion RHorn-backdoor B, notice that if B includes at least $t - 1$ of the x variables, then $|B| \geq t - 1 = s + \log_2 s - 1$, as claimed. Otherwise, B does not contain at least two of the x variables, and we must therefore rename at least one of these two variables, say x_1, to make C_x Horn. This implies that we must flip all variables $y_j \notin B$ because of the clauses $C_z^{1,j}$ which now already have a positive literal, x_1. However, because of the clause C_y, we can flip at most one y variable, and it follows that at least $s - 1$ of the y variables are in B. Moreover, we also have that all $\log_2 s$ of the z variables are in B, because otherwise, irrespective of how the z variables are renamed, in at least one $C_z^{1,j}$ clause a z variable will appear positively, violating the Horn property. Hence, $|B| \geq s - 1 + \log_2 s$, as claimed. This finishes the proof. □

The relationship between the minimum strong and deletion backdoor size w.r.t. Horn and RHorn can be summarized as follows:

$$\text{strong RHorn}^{\{\}} < \text{strong RHorn} < \text{del RHorn} < \text{strong Horn} = \text{del Horn}$$

To understand the justification for the above relationships, recall that a strong Horn-backdoor is also a deletion Horn-backdoor, which is also a deletion RHorn-backdoor. That the third inequality is strict follows from the simple observation that for a RHorn formula F that is not a Horn formula, the smallest deletion RHorn-backdoor is of size zero while the smallest deletion Horn-backdoors (and hence the smallest strong Horn-backdoors) are of a non-zero size. The second of these inequalities follows from Theorem 2, and the first inequality is supported by Proposition 3.

4 Empirical evaluation of backdoor size with different sub-solvers

This section presents an empirical study of the size of backdoor sets in various settings involving a number of static and dynamic sub-solvers. Section 4.1 describes our methodology (i.e., how we compute or approximate a smallest size backdoor set), Section 4.2 discusses the benchmark domains used for the experiments, and Section 4.3 presents the results.

4.1 Computing smallest backdoors

Given a CNF formula, how can we compute a smallest backdoor for it with respect to a given sub-solver? We consider this question w.r.t. the tractable classes Horn and RHorn, as well as w.r.t. polynomial-time sub-solvers that capture the propagation mechanisms employed by DPLL-style SAT solvers.

Recall from Proposition 2 that deciding whether there is a deletion backdoor of size k for a formula F is a problem within NP. Hence, we can simply write, for example, an *integer program* to compute such a backdoor. The known equivalence between deletion and strong backdoors w.r.t. the class Horn [47] implies that we can actually use this procedure to compute the smallest strong Horn-backdoors. On the other hand, Theorem 2, which shows that strong RHorn-backdoors are not equivalent to deletion RHorn-backdoors, implies that finding the minimum deletion RHorn-backdoor size will yield only an upper bound on the minimum size of strong RHorn-backdoors in general, which is what we obtained in our experiments. Similarly, the problem of deciding the existence of a size-k backdoor is not known to be in NP for any of the propagation-based sub-solvers that we consider. Hence, we resorted to indirect techniques for obtaining upper bounds on the minimum size of strong backdoors w.r.t. DPLL-style sub-solvers.

Smallest strong horn-backdoors The problem of finding a smallest strong Horn-backdoor can be formulated as a 0–1 integer programming problem using the equivalence to deletion backdoors [47]. Given a formula F, associate with each Boolean variable x_i a 0–1 variable y_i, where $y_i = 0$ denotes that the corresponding variable x_i is deleted from F (and added to the backdoor). For every clause $c \in F$, let $c^+ = \{i \mid x_i^1 \in c\}$ and $c^- = \{i \mid x_i^0 \in c\}$. The smallest deletion/strong Horn-backdoor problem is formulated as follows:

$$\begin{array}{lll} \text{minimize} & \sum_{i \in \text{var}(F)} (1 - y_i) & \\ \text{subject to} & \sum_{i \in c^+} y_i \leq 1 & \forall c \in F \\ & y_i \in \{0, 1\} & \forall x_i \in \text{var}(F) \end{array}$$

The constraints ensure that each clause is Horn (in each clause, the total number of non-deleted positive literals is at most one). The objective function minimizes the size of the backdoor.

Smallest deletion RHorn-backdoors The problem of finding a smallest deletion RHorn-backdoor can be formulated similarly. Given a formula F, associate with each Boolean variable x_i three 0–1 variables y_{1i}, y_{2i}, y_{3i}, where $y_{1i} = 1$ denotes that x_i is not renamed in F, $y_{2i} = 1$ denotes that x_i is renamed in F, and $y_{3i} = 1$ denotes that x_i is deleted from F (and added to the deletion backdoor). The smallest deletion RHorn-backdoor problem is formulated as follows:

$$\begin{array}{lcl} \text{minimize} & \sum_{i \in \text{var}(F)} y_{3i} & \\ \text{subject to} & y_{1i} + y_{2i} + y_{3i} = 1 & \forall x_i \in \text{var}(F) \\ & \sum_{i \in c^+} y_{1i} + \sum_{i \in c^-} y_{2i} \leq 1 & \forall c \in F \\ & y_{1i}, y_{2i}, y_{3i} \in \{0, 1\} & \forall x_i \in \text{var}(F) \end{array}$$

The first set of constraints ensures that each Boolean variable x_i is either non-renamed, renamed, or deleted. The second set of constraints ensures that each clause is Horn (in each clause, the total number of non-renamed positive literals and renamed negative literals is at most one). The objective function minimizes the size of the backdoor.

We use the above encodings and the IBM ILOG CPLEX libraries [35] for experimenting with Horn- and RHorn-backdoors.

Smallest strong (PROB+PL)-, PROB-, (UP+PL)-, UP-, and PL-backdoors Following previous work [39, 59], we used the complete randomized solver *Satz-Rand* [30] to find small strong backdoors. *Satz-Rand* employs a randomized variable selection heuristic as well as a laborious propagation procedure applied at each search node which includes probing, unit propagation, and pure literal elimination. We modified the *Satz-Rand* code so that we could turn on/off each of the propagation techniques (probing,[11] UP, and PL) while keeping the randomized branch heuristic. This allowed us to consider the sub-solvers $S \in \{\text{PL}^{\{\}}, \text{UP}, \text{UP+PL}, \text{PROB}, \text{PROB+PL}\}$. We obtained an upper bound on the size of the smallest strong S-backdoor by running *Satz-Rand* multiple times with different seeds[12] and recording the set of variables on which the solver branched when proving unsatisfiability.[13] For every possible assignment τ to this set B of variables, *Satz* decided the simplified formula $F[\tau/B]$ in a branch-free manner by applying the propagation mechanism S, hence the size of B is an upper bound on the minimum strong S-backdoor size. For each problem instance and each sub-solver, we recorded the smallest backdoor size found across all runs as an upper bound on the minimum strong backdoor size for this instance–sub-solver combination.

4.2 Benchmark domains

We considered five benchmark domains for our experiments.

Graph coloring This domain encodes the problem of computing a legal k-coloring of a given undirected graph with a given k. The instances were generated using the clique-hiding graph generator of [5]. All of the graphs generated contained a (hidden) clique of size 4, and each remaining possible edge in the graph was included with probability 0.5. All SAT-encoded instances are unsatisfiable when the number of colors is 3. The twelve variables

[11] The probing propagation procedure (PROB) was applied until closure, similarly to UP and PL, that is, until no variable could be determined by probing—as opposed to the default, where *Satz* scans the variables with probing only once.

[12] Within each run, restarts were disabled.

[13] All our benchmark instances are unsatisfiable.

representing color assignments to the four vertices in the hidden 4-clique constitute a strong E-backdoor, since any assignment of colors to these four vertices will fail at least one coloring constraint.

MAP planning (AI planning) This is a synthetic logistics planning domain for which the size of strong UP-backdoors is well understood [33]. In this domain, n is the number of nodes in the map graph and k is the number of locations to visit. All MAP instances considered are unsatisfiable—they encode possible visit plans which are of length one planning step less than the length of the shortest feasible plan. Hoffmann et al. [33] show that MAP instances with $k = 2n - 3$ (called asymmetric) have logarithmic-size DPLL refutations (and backdoors). We evaluated the size of the backdoors in asymmetric MAP instances of various sizes from $n = 5$ to $n = 50$.

Pure Nash equilibrium (Game Theory) These instances encode the problem of computing the existence of an equilibrium strategy [46] in a multi-player graphical game. They were used for the first time in a study of the existence of such equilibria under varying network topology by Dilkina et al. [17]. In a game, interactions between players can be represented by an undirected graph where nodes represent players and edges represent mutual dependencies between players [38]. Each player has a finite set of actions and a payoff function that assigns a real number to every selection of actions by the player and his neighbors. Here we considered binary games, where each player has exactly two available actions. Our focus was on random graphical games where each payoff value is chosen uniformly and independently at random from the interval [0, 1] and the interaction graphs are drawn from the Erdös-Rényi random graph model $\mathbb{G}(n, p)$ [24]. In a pure Nash equilibrium (PNE), each player chooses an action and has no incentive to unilaterally deviate and change his action, since the actions chosen by the other players remain fixed (i.e., each player chooses an action which is the best response to the choices of his neighbors). We encoded the problem of deciding whether a graphical game has a PNE as a CNF formula that is satisfiable if and only if the given game has a PNE. The details of this encoding may be found in the Appendix.

Automotive configuration This is a real-world SAT benchmark whose instances encode problems from the validation and verification of automotive product configuration data for Daimler Chrysler's Mercedes car lines [56]. We considered a set of unsatisfiable instances which is available at http://www-sr.informatik.uni-tuebingen.de/~sinz/DC/.

Bridge faults (Verification) This benchmark set is part of SATLIB [34], where it is listed as BF. This is a real-world SAT benchmark whose instances encode problems from the verification of bridge faults. The instances were generated by Nemesis, a test-pattern generation program for realistic bridge faults in CMOS integrated circuits. We also obtained results on the *ssa0432-003* instance, which is one of the eight instances that test for single-at-stuck faults (also part of the BF benchmark set).

4.3 Strong backdoor size in practice

The complexity of backdoor detection limits the usefulness of backdoors as a solution concept for combinatorial problems. However, the notion of backdoors can be applied as a tool for analyzing and understanding the efficiency of performance of state-of-the-art solvers on many real-world instances. In this section, we demonstrate that although one may not always be able to efficiently identify small backdoor sets, in practice many *real-world* combinatorial problems have surprisingly small dynamic backdoors—and SAT solvers (such as *Satz*, which we employed in our experiments) often detect reasonably small backdoors.

Specifically, in our experimental evaluation we computed upper bounds on the minimum size of deletion Horn- and RHorn-backdoors, and of strong backdoors w.r.t. several sub-solvers in the five problem domains discussed earlier in this section. The results for Graph Coloring, MAP Planning, and Pure Nash Equilibrium are shown in Table 2, and those for Automotive Configuration and Bridge Faults are presented in Table 3.

In solving the integer programming instances to determine minimum deletion Horn- and RHorn-backdoors, the CPLEX solver was allocated a maximum of 20 minutes. The Horn computation was in fact very easy for CPLEX. All instances were solved to optimality in less than 1 second, except for the Pure Nash Equilibrium instances with 3000 and 4000 variables, which took up to 3 seconds. Computing optimal deletion RHorn-backdoor sizes was computationally easy for the Automotive Configuration, MAP Planning, and Pure Nash Equilibrium benchmarks: less than 6 seconds for all Automotive Configuration instances, less than 20 seconds for the MAP instances, and less than 13 seconds for the PNE instances. However, the integer programming instances encoding the problem of finding minimum deletion RHorn-backdoors for the Graph Coloring and Bridge Fault instances were very hard to solve to optimality. Here we report results based on the best feasible solutions found

Table 2 Average strong backdoor size, as a percentage of the number of variables, for various ensembles of instances from Graph Coloring, MAP Planning, and Pure Nash Equilibrium (PNE). For each ensemble, the number of instances averaged over is given in parentheses after the problem name. The RHorn figures are for deletion backdoors. Horn- and RHorn-backdoor sizes are optimal (minimum), except the ones marked with an asterisk; the rest are upper bounds

instance set	num vars	num clauses	Static (opt, %)		Dynamic (upper bound, %)					
			HORN	RHORN	PROB+PL	PROB	UP+PL	UP	$PL^{\{\}}$	E
Graph Coloring										
gcp_100 (5)	300	7,557.7	66.67	*57.73	0.33	0.33	1.43	1.47	34.63	4.00
gcp_200 (5)	600	30,122.0	66.67	*62.47	0.17	0.17	0.73	0.73	30.43	2.00
gcp_300 (5)	900	67,724.4	66.67	*63.82	0.11	0.11	0.51	0.51	27.36	1.33
gcp_400 (5)	1,200	119,997.4	66.67	*64.48	0.08	0.08	0.38	0.40	26.82	1.00
gcp_500 (5)	1,500	187,556.0	66.67	*64.88	0.07	0.07	0.40	0.41	32.75	0.80
AI Planning: MAP										
map_5_7 (1)	249	720	38.96	37.75	0	0	2.41	2.41	62.65	
map_10_17 (1)	1,284	5,000	44.55	44.31	0	0	1.25	1.25	59.27	
map_20_37 (1)	5,754	33,360	47.31	47.25	0	0	0.63	0.63	57.72	
map_30_57 (1)	13,424	103,120	48.21	48.19	0	0	0.42	0.42	57.22	
map_40_77 (1)	24,294	232,280	48.66	48.65	0	0	0.31	0.31	56.97	
map_50_97 (1)	38,364	438,840	48.93	48.92	0	0	0.25	0.25	56.83	
Game Theory: PNE										
pne (20)	2,000	40,958.9	67.88	66.86	0.00	0.00	0.07	0.09	0.22	
pne (20)	3,000	60,039.7	67.66	66.55	0.00	0.00	0.03	0.04	0.08	
pne (20)	4,000	78,839.1	67.97	66.93	0.00	0.00	0.03	0.03	0.07	
pne (20)	5,000	98,930.8	67.81	66.80	0.00	0.00	0.02	0.02	0.05	

Table 3 Average strong backdoor size, as a percentage of the number of variables, for various ensembles of instances from Automotive Configuration and Bridge Faults (BF). For each ensemble, the number of instances averaged over is given in parentheses after the problem name. The RHorn figures are for deletion backdoors. Horn- and RHorn-backdoor sizes are optimal (minimum), except the ones marked with an asterisk; the rest are upper bounds

instance set	num vars	num clauses	Static (optimal, %)		Dynamic (upper bound, %)				
			HORN	RHORN	PROB+PL	PROB	UP+PL	UP	PL
Automotive Configuration									
c168_fw_sz (5)	1,698	5,646.8	14.32	2.83	0.06	0.06	0.45	0.60	4.16
c168_fw_ut (8)	1,909	7,489.3	23.62	5.50	0.00	0.00	0.06	0.06	10.84
c170_fr_sz (5)	1,659	4,989.8	9.98	3.57	0.00	0.00	0.45	0.48	3.40
c202_fs_sz (8)	1,750	6,227.8	12.31	4.55	0.00	0.00	0.21	0.27	4.11
c202_fw_rz/sz (11)	1,799	8,906.9	14.48	6.12	0.02	0.02	0.86	1.06	6.24
c202_fw_ut (2)	2,038	11,352.0	21.25	7.61	0.00	0.00	0.15	0.25	9.13
c208_fA_rz/sz (5)	1,608	5,286.2	10.52	4.51	0.00	0.00	0.32	0.37	2.30
c208_fA_ut (2)	1,876	7,335.5	23.13	7.46	0.00	0.00	0.08	0.08	13.33
c208_fc_rz (2)	1,654	5,567.0	10.28	4.59	0.00	0.00	0.67	0.79	3.20
c208_fc_sz (5)	1,654	5,571.8	10.47	4.68	0.01	0.01	0.40	0.76	3.17
c210_fs_rz (3)	1,755	5,764.3	11.64	4.22	0.00	0.00	0.55	0.68	3.30
c210_fs_sz (10)	1,755	5,796.8	11.77	4.35	0.00	0.00	0.39	0.54	3.51
c210_fw_rz (3)	1,789	7,408.3	12.54	4.81	0.00	0.00	0.65	0.86	4.14
c210_fw_rz/sz (12)	1,789	7,511.8	13.74	5.37	0.02	0.02	0.41	0.73	5.62
c210_fw_ut (2)	2,024	9,720.0	20.73	7.31	0.00	0.00	0.37	0.42	12.80
c220_fv_rz/sz (9)	1,728	4,758.2	9.14	2.92	0.09	0.16	0.33	0.54	5.53
Verification: BF									
bf0432-007 (1)	1,040	3,668	50.10	*28.17	1.54	1.54	11.54	12.69	85.19
bf1355-075 (1)	2,180	6,778	52.06	*26.15	2.80	2.80	5.00	5.00	64.17
bf1355-638 (1)	2,177	6,768	51.68	*25.86	2.07	2.07	6.84	6.84	62.93
bf2670-001 (1)	1,393	3,434	41.92	*20.96	1.22	1.22	1.29	1.65	48.96
ssa0432-003 (1)	435	1,027	48.97	20.46	0.00	0.00	3.91	3.91	88.51

in a limited computational time of 20 minutes, which gives an upper bound on the minimum deletion RHorn-backdoor sizes. For all other instances, we report optimal deletion RHorn-backdoor sizes.

The graph-coloring domain illustrates that both deletion/strong Horn-backdoors and deletion RHorn-backdoors can be significantly larger than backdoors w.r.t. empty-clause detection. Recall that by construction, these instances have an E-backdoor (i.e., a backdoor with respect to only empty-clause detection) of size 12. We note that when using DPLL sub-solvers such as UP, *Satz* found backdoors even smaller than the E-backdoor.

In the MAP Planning domain, the deletion/strong Horn-backdoors and deletion RHorn-backdoors were of comparable size and relatively large (39 to 49 % of the number of variables in the formula); as expected, the strong UP-backdoors were quite small. Interestingly, *Satz* solved these instances without any search at all when using its full propagation procedure. The smallest strong PROB+PL- and PROB-backdoors were of size 0.

For the Game Theory problem instances of computing a pure Nash equilibrium, the deletion/strong Horn-backdoor sets and deletion RHorn-backdoor sets contained approximately 68 and 67 % of the variables, respectively. In contrast, the strong PROB-backdoors were surprisingly small, close to 0 % of the variables.

In the Automotive Configuration problems, while the deletion/strong Horn-backdoors varied between 9 and 24 % of the variables, the RHorn-backdoor sets were considerably smaller, at3 to 8 % of the variables. However, since these are deletion RHorn-backdoor sizes, the minimum strong RHorn-backdoor size could be even smaller. The strong PROB+PL-backdoors involved only 0 to 0.09 % of the variables.

Finally, in the Bridge Faults instances, we can again see that the deletion RHorn-backdoor sets were considerably smaller than the Horn-backdoor sets, involving 20 to 28 % versus 42 to 52 % of the variables. Pure literal elimination was not effective on these problems, resulting in $\text{PL}^{\{\}}$-backdoors of 49 to 89 % of the variables. On the other hand, using probing provided significant gains in backdoor size over unit propagation.

We can conclude that in practice RHorn-backdoors are often substantially smaller than Horn-backdoors, especially in the industrial automotive configuration and bridge fault benchmarks (as opposed to the three synthetic domains). Hence, between these two syntactic tractable classes, RHorn-backdoors are likely to yield significantly more computational advantage for constraint solvers than Horn-backdoors. However, deletion RHorn-backdoors are always considerably larger than backdoors with respect to the DPLL solvers UP, UP+PL, PROB and PROB+PL. In fact, strong PROB- and PROB+PL-backdoors are an order of magnitude smaller than deletion RHorn-backdoors. We found that all benchmarks exhibited strong PROB-backdoors that contained less than 3 % of the variables—providing evidence that real-world SAT instances often posses hidden structure (appropriately defined with respect to effective sub-solvers such as PROB) that succinctly captures the problem combinatorics.

Interestingly, for all benchmark instances, the $\text{PL}^{\{\}}$-backdoor sizes were much larger than the UP- and UP+PL-backdoor sizes, and sometimes larger than the deletion RHorn-backdoor sizes. In particular, adding PL propagation to UP to obtain UP+PL, and adding PL to probing to obtain PROB+PL propagation, had a minor effect on the size of the minimum strong backdoor. In fact, because of the computational overhead in keeping track of pure literals and their minor contribution to improving the effectiveness of backtrack search methods, pure literal elimination has been omitted in state-of-the-art SAT solvers such as *RSat* [51].

5 Backdoor sets in the presence of learning during search

In the remaining part of this work, we discuss the notion of backdoors in the context of learning during search, and the related issue of the order in which various search tree branches are explored. Suppose B is a strong backdoor for F with respect to a certain sub-solver S. By definition, branching on the variables in B is sufficient for a systematic search procedure such as DPLL to solve F using S as the sub-solver. The fact that the systematic search procedure will succeed independent of the order in which it explores various truth valuations of variables in B is, in fact, a crucial notion that has only recently begun to be investigated, in the context of backdoors for mixed-integer programming solvers [19]. In practice, modern SAT solvers such as *RSat* [51], *Minisat* [23], and *zChaff* [45] employ *clause learning* techniques, which allow them to carry over information from previously explored branches to newly considered branches. Prior work on proof methods based on clause learning and the resolution proof system suggests that, especially for unsatisfiable formulas, some variable-value assignment orders may lead to significantly shorter search times than others when information is carried over from one search branch to another. In other words, it is very possible that "learning-sensitive" backdoors are much smaller than "traditional" strong backdoors if appropriate variable-value ordering is used and useful information is carried over from previously explored branches. As we will show through a carefully constructed example, the presence of clause learning can in fact make an exponential difference in the backdoor size. To make the notion of incorporating learning-during-search into backdoor sets more precise, we introduce the following extended definition:

Definition 8 Given a Boolean formula F and a set $B \subseteq var(F)$, B is a *learning-sensitive backdoor* for F w.r.t. a sub-solver S if there exists a search-tree exploration order such that a clause-learning SAT solver branching only on the variables in B, with this order and with S as the sub-solver at the leaves of the search tree, either finds a satisfying assignment for F or proves that F is unsatisfiable.

Note that, as before, each leaf of this search tree corresponds to a truth assignment $\tau : B \to \{0, 1\}$ and induces a simplified formula $F[\tau/B]$ to be solved by S. However, the tree search is naturally allowed to carry over and use learned information from previous branches in order to help S determine $F[\tau/B]$. Thus, while $F[\tau/B]$ may not be solvable by S *per se*, additional information gathered from previously explored branches may help S solve $F[\tau/B]$. Definition 8 captures the traditional notions of both weak and strong backdoor sets, in that there is a critical set of variables which finds a satisfying assignment if one exists, and shows infeasibility otherwise. We explain the power of learning sensitivity through the following example of an unsatisfiable formula, which has a natural learning-sensitive UP-backdoor (i.e., a learning-sensitive backdoor with respect to the unit propagation subsolver) of size 1 although its smallest traditional strong UP-backdoor is of size 2. We will then generalize this observation to an exponential separation between the power of learning-sensitive backdoors and traditional strong backdoors for SAT.

Example 2 Consider the following unsatisfiable SAT instance F_1:

$$\begin{aligned} F_1 = {} & (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee q) \wedge (\neg x \vee q \vee r) \wedge \\ & (\neg q \vee a) \wedge (\neg q \vee \neg a \vee b) \wedge (\neg q \vee \neg a \vee \neg b) \wedge \\ & (\neg r \vee a) \wedge (\neg r \vee \neg a \vee b) \wedge (\neg r \vee \neg a \vee \neg b) \end{aligned}$$

We claim that $\{x\}$ is a size-1 learning-sensitive UP-backdoor for F_1, while all traditional strong UP-backdoors are of size at least two. First, let us understand why $\{x\}$ works as a backdoor set when clause learning is allowed. When we set $x = 0$, this implies—by unit propagation—the literals p_1 and p_2. Together, they imply q, which in turn implies a. Finally, q and a together imply both b and $\neg b$, causing a contradiction. At this point, a clause-learning algorithm will realize that the literal q forms what is called a unique implication point (UIP) for this conflict, and will learn the singleton clause $\neg q$.[14] Now, when we set $x = 1$, this, along with the learned clause $\neg q$, will unit propagate one of the clauses of F_1 and imply r, which will then imply a and cause a contradiction as before. Thus, setting $x = 0$ leads to a contradiction by unit propagation as well as a learned clause, and setting $x = 1$ after this also leads to a contradiction.

[14]We omit the details of the 1-UIP clause-learning mechanism and refer the interested reader to [45].

To see that there is no traditional strong UP-backdoor of size one (and, in particular, that $\{x\}$ does not work as a strong UP-backdoor without the help of the learned clause $\neg q$), observe that for every variable of F_1, there exists at least one polarity in which it does not appear in any 1- or 2-clause (i.e., a clause containing only 1 or 2 variables) and therefore there is no empty clause generation or unit propagation under at least one truth assignment for that variable. (Note that F_1 does not have any 1-clauses to begin with.) For example, q does not appear in any 2-clause of F_1 and therefore setting $q = 0$ does not cause any unit propagation at all, eliminating any chance of deducing a conflict. Similarly, setting $x = 1$ does not cause any unit propagation. In fact, no variable of F_1 can lead to a contradiction by itself (under *either* truth assignment to it), hence F_1 has no traditional strong UP-backdoor of size 1. Note that $\{x, q\}$ is a traditional strong UP-backdoor of size two for F_1.

Theorem 3 *There are* unsatisfiable *formulas for which the smallest learning-sensitive UP-backdoors are exponentially smaller than the smallest traditional strong UP-backdoors.*

Proof We, in fact, provide two proofs of this statement by constructing two unsatisfiable formulas F_2 and F_3 over $N = k + 3 \cdot 2^k$ variables and $M = 4 \cdot 2^k$ clauses, with the following property: Both formulas have a learning-sensitive backdoor of size $k = \Theta(\log N)$ but no traditional strong backdoor of size smaller than $2^k + k = \Theta(N)$. The formula F_2 is perhaps a bit easier to understand and has a relatively weak ordering requirement for the size-k learning-sensitive backdoor to work (namely, that the all-1's truth assignment must be evaluated at the very end); the formula F_3, on the other hand, requires a strict value ordering to work as a backdoor (namely, the lexicographic order from $000\ldots0$ to $111\ldots1$) and highlights the strong role a good branching order plays in the effectiveness of backdoors.

Let σ denote a string of k 0's and 1's, such as $101\ldots0$. This naturally corresponds to a truth assignment to k Boolean variables. The variables of both F_2 and F_3 will be

$$\{x_i \mid 1 \le i \le k\} \bigcup \{q_\sigma, a_\sigma, b_\sigma \mid \sigma \text{ is a 0–1 string of length } k\}$$

For a string σ, consider the unique clause C_σ over the k variables x_i that falsifies σ. For example, for $k = 3$ and $\sigma = 001$, the clause C_{001} is $(x_1 \vee x_2 \vee \neg x_3)$. One of the strings, without loss of generality the all-1's string $111\ldots1$, will play a special role below, and for succinctness we denote it in bold type as $\mathbf{1}$. The formula F_2 is defined as follows:

$$F_2 = \bigwedge_{\sigma \neq \mathbf{1}} ((C_\sigma \vee q_\sigma) \wedge (\neg q_\sigma \vee a_\sigma) \wedge (\neg q_\sigma \vee \neg a_\sigma \vee b_\sigma) \wedge (\neg q_\sigma \vee \neg a_\sigma \vee \neg b_\sigma))$$
$$\bigwedge \left(\left(C_{\mathbf{1}} \vee \bigvee_{\sigma'} q_{\sigma'} \right) \wedge (\neg q_{\mathbf{1}} \vee a_{\mathbf{1}}) \wedge (\neg q_{\mathbf{1}} \vee \neg a_{\mathbf{1}} \vee b_{\mathbf{1}}) \wedge (\neg q_{\mathbf{1}} \vee \neg a_{\mathbf{1}} \vee \neg b_{\mathbf{1}}) \right)$$

Notice that the sets of clauses in F_2 corresponding to the $2^k - 1$ strings $\sigma \neq \mathbf{1}$ are similar in structure and involve distinct variables other than the x variables. The first clause corresponding to the string $\mathbf{1}$ is, however, much longer—it includes *all* 2^k of the q variables, along with all k of the x variables. The formula F_3 is similar to it, the only difference being that if we think of the 2^k strings σ as being ordered lexicographically, then the first clause for σ includes the q variables for all the preceding σ's. More precisely, using $\preceq$ for the lexicographic order over the strings σ, we have

$$F_3 = \bigwedge_{\sigma} \left(\left(C_\sigma \vee \bigvee_{\sigma' \preceq \sigma} q_{\sigma'} \right) \wedge (\neg q_\sigma \vee a_\sigma) \wedge (\neg q_\sigma \vee \neg a_\sigma \vee b_\sigma) \wedge (\neg q_\sigma \vee \neg a_\sigma \vee \neg b_\sigma) \right)$$

We claim that $B = \{x_i \mid 1 \leq i \leq k\}$ forms a learning-sensitive UP-backdoor for both F_2 and F_3. To see this, consider the all-0's assignment to the variables in B. This assignment falsifies $C_{000\ldots0}$ and by unit propagation implies $q_{000\ldots0}$ in both F_2 and F_3. As in Example 2, this implies $a_{000\ldots0}$ and eventually leads to a contradiction. At this point, the search procedure, again as in that example, learns the singleton 1-UIP clause $\neg q_{000\ldots0}$. Now, consider the next string in the lexicographic order, namely, $000\ldots1$, which falsifies $C_{000\ldots1}$. In F_2, this immediately implies $q_{000\ldots1}$, while in F_3, this along with the learned clause $\neg q_{000\ldots0}$ implies $q_{000\ldots1}$. In either case, we continue as before, derive a contradiction, and learn the singleton clause $\neg q_{000\ldots1}$. The process continues until we arrive at the last string, $\sigma = \mathbf{1}$. Now both F_2 and F_3 use *all* of the $2^k - 1$ clauses learned so far in order to deduce $q_{\mathbf{1}}$ and thus derive a contradiction. This completes the proof that the set B forms a learning-sensitive backdoor for both F_2 and F_3.

Note that the use of learned clauses was crucial in the above refutations. In the case of traditional strong backdoors, we do not have access to learned clauses. To see that all traditional strong backdoors for F_2 and F_3 will need at least $2^k + k$ variables, we make use of the observation that these formulas are minimally unsatisfiable, that is, removing any single clause turns them into satisfiable formulas.[15] It follows that any proof of unsatisfiability must "make use" of all clauses of these formulas. Specifically, consider the longest clause in both F_2 and F_3, $C_{\text{long}} \equiv (C_{\mathbf{1}} \vee \bigvee_{\sigma'} q_{\sigma'})$, which involves $2^k + k$ variables. Consider a partial truth assignment τ that satisfies all clauses of the formula other than C_{long}; τ exists because of the formula being minimally unsatisfiable. It follows that for any traditional backdoor set B, the truth assignment $\tau|_B$ consistent with σ cannot cause any unit propagation through clauses other than C_{long}, nor can it generate an empty clause and deduce a conflict through these other clauses themselves. To prove the formula unsatisfiable under truth assignment τ, the sub-solver must therefore deduce that clause C_{long} is violated. Without access to previously learned clauses, all literals of C_{long} must be negated by either being included in B and assigned truth values, or because of being implied by unit propagation of other clauses—which we argued does not happen for τ. Thus, we conclude that all variables of C_{long} must belong to B, implying $|B| \geq 2^k + k$ as claimed. $\square$

In fact, the discussion in the proof of Theorem 3 also reveals that for the constructed formula F_3, any value ordering that starts by assigning 1's to all x_i's will lead to a learning-sensitive backdoor of size no smaller than 2^k. This immediately yields the following result, underscoring the importance of the "right" value ordering even among various learning-sensitive backdoors.

Corollary 1 *There are* unsatisfiable *formulas for which one value ordering of the variables can lead to exponentially smaller learning-sensitive UP-backdoors than a different value ordering.*

We now turn our attention to the study of strong backdoors for *satisfiable* instances. We show that clause learning not only helps with unsatisfiable instances, it can also lead

[15]To see that F_2 is minimally unsatisfiable, consider removing a clause D from F_2. If D is one of the clauses involving C_σ for some σ, set each x_i to σ_i, set all q variables to 0, and set all a and b variables arbitrarily. Otherwise, D must be one of the clauses involving $\neg q_\sigma$, a_σ, and possibly b_σ; in this case, set x_i to σ_i as before, set q_σ to 1, set all remaining q variables to 0, set a_σ and b_σ to satisfy the two remaining clauses that $\neg q_\sigma$ appears in (this can always be done), and set the remaining a and b variables arbitrarily. The resulting truth assignment satisfies F_2. The same construction works for F_3.

to (somewhat) smaller strong backdoors for satisfiable instances. In fact, our experiments suggest a much more drastic impact of clause learning on backdoors for practical satisfiable instances than on backdoors for unsatisfiable instances.

Theorem 4 *There are* satisfiable *formulas for which there exist learning-sensitive UP-backdoors that are smaller than the smallest traditional strong UP-backdoors.*

Proof Consider the satisfiable variant F_1^{SAT} of the $F1$ instance from Example 2 obtained by replacing the last clause, $(\neg r \vee \neg a \vee \neg b)$, with the clause $(\neg r \vee p_1)$:

$$\begin{aligned} F_1^{\text{SAT}} = {} & (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee q) \wedge (\neg x \vee q \vee r) \wedge \\ & (\neg q \vee a) \wedge (\neg q \vee \neg a \vee b) \wedge (\neg q \vee \neg a \vee \neg b) \wedge \\ & (\neg r \vee a) \wedge (\neg r \vee \neg a \vee b) \wedge (\neg r \vee p_1) \end{aligned}$$

It is easy to see that F_1^{SAT} is satisfiable, with $(x, p_1, p_2, q, r, a, b) = (1, 1, 0, 0, 1, 1, 1)$ being the only solution. As before, for every variable of F_1^{SAT}, there exists at least one polarity in which it does not occur in any 1- or 2-clause. Furthermore, no literal appears in all clauses, and so satisfying just one particular literal cannot make the unit propagation sub-solver conclude that F_1^{SAT} is satisfied by the current partial truth assignment. Putting these two observations together, we have that fixing the value of a variable to one polarity does not cause any unit propagation and does not automatically satisfy all clauses. Therefore, this variable by itself is not a traditional strong backdoor for F_1^{SAT}.

On the other hand, $\{x\}$ again forms a learning-sensitive backdoor for this formula. To see this, note that when $x = 0$, we deduce a conflict as before for F_1 and learn the singleton clause $\neg q$. When x is set to 1, this, together with the learned clause $\neg q$, implies r, which in turn implies a and then b. r also implies p_1, which, together with $\neg q$, implies $\neg p_2$. At this point, all variables have truth values assigned by unit propagation and all clauses are satisfied. Thus, $\{x\}$ correctly serves as a learning-sensitive backdoor for F_1^{SAT}. □

As a closing remark, we note that the presence of clause learning does not affect the power of weak backdoors w.r.t. tractable SAT sub-classes that are closed under clause removal (see Definition 5) such as 2CNF, Horn, or RHorn. Since backdoors w.r.t. such syntactic sub-solvers capture "static" substructure, they fail to benefit from clause learning. This is formalized in the proposition below. Whether clause learning can reduce weak backdoor size w.r.t. dynamic sub-solvers remains open. No such reduction exists for the formula F_1^{SAT} used in the proof of Theorem 4, as it does have a weak backdoor of size 1, namely $\{r\}$ with truth assignment 1.

Proposition 4 *Clause learning does not reduce the size of weak backdoors with respect to any syntactic class C that is closed under clause removal.*

Proof Let F be a CNF formula for which B is a learning-sensitive C-backdoor. Let τ be the final truth assignment to B explored by the clause-learning SAT solver before determining that the resulting simplified formula is in C. Specifically, the solver has determined that the resulting formula F', consisting of $F[\tau/B]$ along with all clauses learned so far and restricted by τ, is in the class C. However, since C is closed under clause removal, this means that $F[\tau/B]$ itself belongs to C. Hence, B is also a traditional weak C-backdoor for F, as witnessed by the truth assignment τ. □

6 Experimental results: the effect of clause learning on backdoor size

In order to experimentally evaluate the effect of clause learning on the size of backdoors, we computed upper bounds on size of the smallest learning-sensitive UP-backdoors and of traditional UP-backdoors. We used some well-known SAT instances from SATLIB [34].

To compute the size of (not necessarily minimal) learning-sensitive backdoors, we used the SAT solver *RSat* [51]. At every search node, *RSat* employs unit propagation as the simplification procedure. At every conflict, it employs conflict clause learning based on a so-called unique implication point (UIP) scheme. We turned off restarts, and randomized the variable- and value-selection heuristics in order to force *RSat* to explore different search directions during multiple runs. In addition, we added code to trace the set of variables used for branching during search, that is, the backdoor set implicitly used by the solver. We ran this modified version of *RSat* 5,000 times per instance and recorded the smallest backdoor set encountered among all runs.

While *RSat* is mainly geared to utilizing clause learning, it might not be very good at finding small sets of branching variables. On the other hand, the SAT solver *Satz*, which does not use clause learning, relies heavily on good variable-selection heuristics in order to minimize the search time. Hence *Satz* is better at discovering smaller branching sets than *RSat*. For this reason, we used a modified version of *Satz-Rand* [30, 42] that uses unit propagation as a sub-solver and also traces the set of branch variables. We ran the modified *Satz* 5000 times per instance and recorded the smallest backdoor set among all runs. This gave us an upper bound on the actual minimum weak backdoor size for satisfiable instances and on the minimum strong backdoor size for unsatisfiable instances. Using *Satz* instead of a modified version of *RSat* with learning turned off gave us much better bounds on traditional UP-backdoors.

Table 4 Upper bounds on the minimum backdoor size when using only unit propagation (within Satz), in column ‘without CL’, and when using clause learning and unit propagation (within RSat), in column ‘with CL’ . Results are given as a percentage of the number of variables. Bold font highlights clause learning results that are at least twofold better than the results without CL

Instance				UP-backdoor size	
Name	Satisfiable	num vars	num clauses	without CL	with CL
bf0432-007	no	1,040	3,668	12.69 %	12.12 %
bf1355-075	no	2,180	6,778	5.00 %	3.90 %
bf1355-638	no	2,177	6,768	6.84 %	3.86 %
bf2670-001	no	1,393	3,434	1.65 %	1.22 %
apex7_gr_2pin_w4	no	1,322	10,940	20.73 %	12.25 %
parity_unsat_4_5	no	2,508	17,295	39.07 %	**9.85 %**
anomaly	yes	48	261	4.17 %	4.17 %
medium	yes	116	953	14.66 %	**1.72 %**
huge	yes	459	7,054	3.27 %	**1.09 %**
bw_large.a	yes	459	4,675	3.49 %	**1.53 %**
bw_large.b	yes	1,087	13,772	11.59 %	**1.93 %**
bw_large.c	yes	3,016	50,457	13.76 %	**2.95 %**
bw_large.d	yes	6,325	131,973	43.27 %	**3.37 %**

The results are summarized in Table 4. Among the unsatisfiable instances (shown in the upper half of the table), the upper bounds on the learning-sensitive backdoors, although sometimes a factor of 1.5 or 2 smaller, are typically not drastically better from those obtained for traditional backdoors. A notable exception, however, is the *parity* instance where including clause learning reduces the backdoor upper bound from over 39 % of all problem variables to under 10 %. The reduction in backdoor size is significantly more apparent amongst the satisfiable instances (shown in the lower half of the table). For example, on the AI planning instance *bw_large.d*, the upper bound found by the SAT solvers on the backdoor size reduces from over 43 % of all variables to under 3.5 % when a clause-learning heavy solver such as *RSat* is used. Similarly, the upper bound on the backdoor size for the SATLIB instance named *medium* reduces from nearly 15 % to under 2 % when learning-sensitive backdoors are used. Overall, these experiments provide practical evidence in support of our theoretical findings that learning-sensitive backdoors can often be much smaller than traditional backdoors.

7 Conclusion

The presence of tractable structure in many real-world instances of combinatorial problems plays a critical role in extending the reach of state-of-the-art constraint solvers to solve these problems. This work explores such structure in the form of backdoor sets, focusing in particular on the tradeoffs between static properties (e.g., tree-width of the constraint graph, simplification to 2CNF, simplification to Horn form) vs. dynamic properties (e.g., simplification to RHorn, propagation using UP or PL, probing, inconsistency detection) exploited by the prevalent notions of structure.

The complexity of finding backdoors is influenced significantly by the features of the underlying sub-solver or tractable problem class. In particular, while the problem of determining whether there exists a strong Horn- or 2CNF-backdoor (a static class) of size k is known to be in NP and fixed parameter tractable, we showed that this problem becomes harder than NP (unless NP = coNP) as soon as the seemingly small but dynamic feature of empty-clause detection (present in all modern SAT solvers) is incorporated into the classes Horn and 2CNF. While such a feature increases the worst-case complexity of finding backdoors (again, unless NP = coNP), our experiments showed that in practice it also has a clear positive impact: It reduces the size of the backdoors dramatically. For the class RHorn, we proved that deletion backdoors can be exponentially larger than strong backdoors, in contrast to the known results for 2CNF- and Horn-backdoors. Nonetheless, we showed experimentally that deletion RHorn-backdoors themselves can be substantially smaller than strong Horn-backdoors. We also demonstrated that strong backdoors w.r.t. dynamic simplification techniques such as UP, PL, UP+PL, and PROB can be substantially smaller than strong Horn-backdoors and deletion RHorn-backdoors, and that *Satz-Rand* is remarkably good at finding small strong backdoors on a range of unsatisfiable problem domains.

We also extended the concept of backdoors to the context of learning, where information learned from previous search branches is allowed to be used by the sub-solver underlying the backdoor. We proved that the smallest backdoors for SAT that take into account clause learning can be exponentially smaller than traditional strong backdoors oblivious to this critical solver feature. Our empirical results demonstrated that the added power of learning-sensitive backdoors is also often observed in practice.

Acknowledgments This work was supported by the Intelligent Information Systems Institute at Cornell University (AFOSR grant FA9550-04-1-0151), NSF Expeditions in Computing award for Computational Sustainability (grant 0832782), and NSF IIS award (grant 0514429). The first author was partially supported by the NSERC PGS Fellowship. Most of this work was done while the first and last author were at Cornell University.

The authors would like to thank Jörg Hoffmann for providing the generator for the MAP domain. A preliminary version of this article appeared at the 13th International Conference on Principles and Practice of Constraint Programming, Providence, RI in September 2007 [16]. Part of this work was also presented at the (unarchived) 10th International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, FL in January 2008 [18]. Some of the latter parts of this work appeared, without proofs, at the 12th International Conference on Theory and Applications of Satisfiability Testing, Swansea, UK, in June 2009 [20].

Appendix: CNF encoding of the pure nash equilibria problem

For completeness, we describe the Pure Nash Equilibria (PNE) setting in game theory and a translation of instances of this problem to CNF formulas. Consider an n-player game $\mathscr{G}$ in which the actions of a player and his payoff depend on the actions of a subset of the other players. $\mathscr{G}$ is called a *graphical game*; when the payoffs of all players depend on the actions of all other players, the graphical game degenerates into a classical "full-interaction" game. We can represent the mutual interactions between the players in $\mathscr{G}$ as edges in an undirected graph G whose nodes correspond to the players. Specifically, the payoff of a player p is a function of his action and the actions of all his neighbors $Nbr(p)$ in G. This payoff function defines p's *best-response strategy*, which is a mapping from the actions of the players in $Nbr(p)$ to an action for p that maximizes his own payoff. For simplicity, one often assumes that p always has a *unique* best-response action given the actions of his neighbors, and thus talk of the best-response *function*. Although we make this assumption in the instances chosen for our experiments, the CNF translation discussed below applies also to the general case where a player can have several best-response actions in any given setting.

We will assume that each player has a finite set of actions and a payoff function that assigns a real number to every selection of actions by him and his neighbors. For the simplified case of binary games, where each player has exactly two action choices, the CNF encoding discussed below can be naturally simplified so that there is only one Boolean variable per player, the two values of which denote the two possible actions of the player. Our experiments in this work involve only binary games, and hence we use this simplified encoding for the experiments. For completeness, however, we discuss the general encoding here, allowing each player to potentially have a choice between several (but finitely many) possible actions.

In a pure Nash equilibrium, each player chooses an action and has no incentive to unilaterally deviate and change his action, given the actions chosen by the other players remain fixed (i.e., each player has chosen a best response action to the choices of his neighbors). We encode the problem of deciding whether $\mathscr{G}$ has a PNE as a CNF formula F that is satisfiable if and only if $\mathscr{G}$ has a PNE.

For every player p in $\mathscr{G}$ and every possible action a of p, there is a Boolean variable x_p^a in F encoding the choice of action a for p. We add a constraint stating that exactly one of all possible actions of p must be taken.[16] Further, let the neighbors of p be $(q_1, q_2, \dots, q_k)$, in some arbitrary but fixed order. As discussed above, the payoffs in $\mathcal{G}$ determine a set of best-response actions $BR_p(a_1, a_2, \dots, a_k)$ for p when the action of player q_i is a_i, $1 \le i \le k$. Therefore, F must encode the fact that when literals $x_{q_i}^{a_i}$ are True for all $1 \le i \le k$, then literals x_p^b are False for all non-best-response actions of p, i.e., for all $b \in \text{actions}(p) \setminus BR_p(a_1, a_2, \dots, a_k)$.

Overall, these constraints can be summarized as the following set of constraints in F for each player p:

$$\bigvee_{a \in \text{actions}(p)} x_p^a \tag{1}$$

$$\neg x_p^a \vee \neg x_p^{a'} \qquad \forall a, a' \in \text{actions}(p);\ a \neq a' \tag{2}$$

$$\left(\bigwedge_{q_i \in Nbr(p)} x_{q_i}^{a_i} \right) \rightarrow \neg x_p^b \qquad \forall a_i \in \text{actions}(q_i), b \in \text{actions}(a) \setminus BR_p(a_1, a_2, \dots) \tag{3}$$

The last of these constraints is not a clause but can be easily translated into the standard clausal form:

$$\left(\bigvee_{q_i \in Nbr(p)} \neg x_{q_i}^{a_i} \right) \vee \neg x_p^b \qquad \forall a_i \in \text{actions}(q_i), b \in \text{actions}(a) \setminus BR_p(a_1, a_2, \dots) \tag{4}$$

This finishes the translation of the PNE problem on the game $\mathcal{G}$ into a CNF formula F.

References

1. Aspvall, B.: Recognizing disguised NR(1) instances of the satisfiability problem. J. Algoritm. **1**(1), 97–103 (1980)
2. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Inf. Process. Lett. **8**(3), 121–123 (1979)
3. Beame, P., Kautz, H., Sabharwal, A.: Understanding and harnessing the potential of clause learning. JAIR: J. Artif. Intell. Res. **22**, 319–351 (2004)
4. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using SAT procedures instead of BDDs. In: DAC-99: 36th Design Automation Conference, pp. 317–320 (1999)
5. Brockington, M., Culberson, J. C.: Camouflaging independent sets in quasi-random graphs. In: Johnson, D.S., Trick, M.A. (eds.) Cliques, Coloring and Satisfiability: The Second DIMACS Implementation Challenge, volume 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 75–88. American Mathematical Society (1996)
6. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. **98**, 142–170 (1992)
7. Chandru, V., Hooker, J.N.: Detecting embedded Horn structure in propositional logic. Inf. Process. Lett. **42**(2), 109–111 (1992)
8. Chen, H., Dalmau, V.: Beyond hypertree width: decomposition methods without decompositions. In: CP-05: 11th International Conference on Principles and Practice of Constraint Programming, pp. 167–181 (2005)
9. Chen, H., Gomes, C.P., Selman, B.: Formal models of heavy-tailed behavior in combinatorial search. In: CP-01: 7th International Conference on Principles and Practice of Constraint Programming, volume 2239 of LNCS. (2001)

[16]We could, in principle, relax this constraint and require that at least one of the actions must be chosen. This would still maintain the requirement that F is satisfiable if and only if $\mathcal{G}$ has a PNE. However, choosing exactly one action for each player is more natural.

10. Davis, M., Putnam, H.: A computing procedure for quantification theory. Commun. ACM **7**, 201–215 (1960)
11. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Commun. ACM **5**, 394–397 (1962)
12. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers Inc. (2003). ISBN 1558608907
13. Dechter, R., Pearl, J.: Network-based heuristics for constraint-satisfaction problems. Artif. Intell. **34**(1), 1–38 (1987)
14. del Val, A.: On 2-SAT and renamable horn. In: AAAI-00: 17th National Conference on Artificial Intelligence, pp. 279–284 (2000)
15. Deville, Y., Van Hentenryck, P.: An efficient arc consistency algorithm for a class of CSP problems. In: IJCAI-91: 12th International Joint Conference on Artificial Intelligence, pp. 325–330 (1991)
16. Dilkina, B., Gomes, C.P., Sabharwal, A.: Tradeoffs in the complexity of backdoor detection. In: CP-07: 13th International Conference on Principles and Practice of Constraint Programming, volume 4741 of LNCS, pp. 256–270 (2007)
17. Dilkina, B., Gomes, C.P., Sabharwal, A.: The impact of network topology on pure Nash equilibria in graphical games. In: AAAI-07: 22nd Conference on Artificial Intelligence, pp. 42–49 (2007)
18. Dilkina, B., Gomes, C.P., Sabharwal, A.: Tradeoffs in backdoors: inconsistency detection, dynamic simplification, and preprocessing. In: ISAIM-08: 10th International Symposium on Artificial Intelligence and Mathematics (2008)
19. Dilkina, B., Gomes, C.P., Malitsky, Y., Sabharwal, A., Sellmann, M.: Backdoors to combinatorial optimization: feasibility and optimality. In: CPAIOR-09: 6th International Conference on Integration of AI and OR Techniques in Constraint Programming, pp. 56–70 (2009)
20. Dilkina, B., Gomes, C.P., Sabharwal, A.: Backdoors in the context of learning. In: SAT-09: 12th International Conference on Theory and Applications of Satisfiability Testing, volume 5584 of LNCS, pp. 73–79 (2009)
21. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional horn formulae. J. Log. Program. **1**(3), 267–284 (1984)
22. Downey, R.G., Fellows, M.R.: Parameterized Complexity (Monographs in Computer Science). Springer (1998). ISBN 978-0387948836
23. Eén, N., Sörensson, N.: MiniSat: a SAT solver with conflict-clause minimization. In: SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing (2005)
24. Erdos, P., Renyi, A.: On random graphs. Publicationes Mathemticae (Debrecen) **6**, 290–297 (1959)
25. Fichte, J.K., Szeider, S.: Backdoors to normality for disjunctive logic programs. In: AAAI-13: 27th Conference on Artificial Intelligence (2013)
26. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006). ISBN 9783642067570
27. Freuder, E.C.: A sufficient condition for backtrack-free search. J. ACM **29**(1), 24–32 (1982)
28. Freuder, E.C.: A sufficient condition for backtrack-bounded search. J. ACM **32**(4), 755–761 (1985)
29. Freuder, E. C.: Complexity of k-tree structured constraint satisfaction problems. In: AAAI-90: 8th National Conference on Artificial Intelligence, pp. 4–9 (1990)
30. Gomes, C.P., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. In: AAAI-98: 15th National Conference on Artificial Intelligence, pp. 431–437 (1998)
31. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Autom. Reason. **24**(1–2), 67–100 (2000)
32. Henschen, L., Wos, L.: Unit refutations and Horn sets. J. ACM **21**, 590–605 (1974)
33. Hoffmann, J., Gomes, C., Selman, B.: Structure and problem hardness: Goal asymmetry and DPLL proofs in SAT-based planning. Logical Methods Comput. Sci. **3**(1–6), 1–41 (2007)
34. Hoos, H.H., Stützle, T.: SATLIB: an online resource for research on SAT. In: Gent, I.P., van Maaren, H., Walsh, W (eds.) SAT2000, pp. 283–292. IOS Press (2000). http://www.satlib.org
35. ILOG, SA. CPLEX 10.1 Reference Manual (2006)
36. Kautz, H.A., Selman, B.: Planning as satisfiability. In: ECAI-92: 10th European Conference on Artificial Intelligence, pp. 359–363 (1992)
37. Kautz, H.A., Selman, B.: Pushing the envelope: planning, propositional logic, and stochastic search. In: AAAI-96: 13th Conference on Artificial Intelligence, pp. 1194–1201 (1996)
38. Kearns, M.J., Littman, M.L., Singh, S.P.: Graphical models for game theory. In: UAI-01: 17th Conference on Uncertainty in Artificial Intelligence, pp. 253–260 (2001)
39. Kilby, P., Slaney, J.K., Thiébaux, S., Walsh, T.: Backbones and backdoors in satisfiability. In: AAAI-05: 20th National Conference on Artificial Intelligence, pp. 1368–1373 (2005)
40. Kottler, S., Kaufmann, M., Sinz, C.: Computation of renameable horn backdoors. In: SAT-08: 11th International Conference on Theory and Applications of Satisfiability Testing, pp. 154–160 (2008)
41. Lewis, H.R.: Renaming a set of clauses as a Horn set. J. ACM **25**(1), 134–135 (1978)

42. Li, C.M., Anbulagan: Heuristics based on unit propagation for satisfiability problems. In: IJCAI-97: 15th International Joint Conference on Artificial Intelligence, pp. 366–371 (1997)
43. Lynce, I., Marques-Silva, J.P.: Hidden structure in unsatisfiable random 3-SAT: an empirical study. In: ICTAI-06: 16th IEEE International Conference on Tools with Artificial Intelligence, pp. 246–251 (2004)
44. Marques-Silva, J.P., Sakallah, K.A.: GRASP–a new search algorithm for satisfiability. In: ICCAD-96: International Conference on Computer Aided Design, pp. 220–227 (1996)
45. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of DAC-01: 38th Design Automation Conference, pp. 530–535 (2001)
46. Nash, J.: Noncooperative games. Ann. Math. **54**, 289–295 (1951)
47. Nishimura, N., Ragde, P., Szeider, S.: Detecting backdoor sets with respect to Horn and binary clauses. In: SAT-04: 7th International Conference on Theory and Applications of Satisfiability Testing (2004)
48. Nishimura, N., Ragde, P., Szeider, S.: Solving #SAT using vertex covers. In: SAT-06: 9th International Conference on Theory and Applications of Satisfiability Testing, pp. 396–409 (2006)
49. Paris, L., Ostrowski, R., Siegel, P., Sais, L.: Computing Horn strong backdoor sets thanks to local search. In: ICTAI-06: 17th IEEE International Conference on Tools with Artificial Intelligence, pp. 139–143 (2006)
50. Pfandler, A., Rümmele, S., Szeider, S.: Backdoors to abduction. In: IJCAI: International Joint Conference on Artificial Intelligence, pp. 1046–1052 (2013)
51. Pipatsrisawat, K., Darwiche, A.: RS at 2.0: SAT solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA (2007)
52. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers with restarts. In CP-09: 15th International Conference on Principles and Practice of Constraint Programming, volume 5732 of LNCS, pp. 654–668 (2009)
53. Razgon, I., O'Sullivan, B.: Almost 2-sat is fixed-parameter tractable. J. Comput. Syst. Sci. **75**(8), 435–450 (2009)
54. Samer, M., Szeider, S.: Constraint satisfaction with bounded treewidth revisited. In: CP-06: 12th International Conference on Principles and Practice of Constraint Programming, pp. 499–513 (2006)
55. Samer, M., Szeider, S.: Backdoor sets of quantified boolean formulas. J. Autom. Reason. **42**(1), 77–97 (2009)
56. Sinz, C., Kaiser, A., Küchlin, W.: Formal methods for the validation of automotive product configuration data. Artif. Intell. Eng. Des, Anal. Manuf. **17**(1), 75–97 (2003). Special issue on configuration
57. Szeider, S.: Backdoor sets for DLL subsolvers. J. Autom. Reason. **35**(1–3), 73–88 (2005)
58. van Beek, P., Dechter, R.: On the minimality and global consistency of row-convex constraint networks. J. ACM **42**(3), 543–561 (1995)
59. Williams, R., Gomes, C., Selman, B.: Backdoors to typical case complexity. In: IJCAI-03: 18th International Joint Conference on Artificial Intelligence, pp. 1173–1178 (2003)
60. Williams, R., Gomes, C., Selman, B.: On the connections between heavy-tails, backdoors, and restarts in combinatorial search. In SAT-03: 6th International Conference on Theory and Applications of Satisfiability Testing, pp. 222–230 (2003)
61. Zhang, L., Madigan, C.F., Moskewicz, M.H., Malik, S.: Efficient conflict driven learning in a Boolean satisfiability solver. In: ICCAD-01: International Conference on Computer Aided Design, pp. 279–285 (2001)