[under review] manuscript No.
(will be inserted by the editor)

Tradeoffs in the Complexity of Backdoor Detection for Combinatorial Problems

Bistra Dilkina · Carla P. Gomes · Ashish Sabharwal

the date of receipt and acceptance should be inserted later

Abstract There has been considerable interest in the identification of structural properties of combinatorial problems that lead to efficient algorithms for solving them. Some of these properties are "easily" identifiable, while 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. In particular, it was recently shown that the problem of identifying a strong Horn- or 2CNF-backdoor can be solved by exploiting equivalence with deletion backdoors, and is NP-complete. We prove that strong backdoor identification becomes harder than NP (unless NP=coNP) as soon as the inconsequential sounding feature of empty clause detection (present in all modern SAT solvers) is added. More interestingly, in practice such a feature as well as polynomial time constraint propagation mechanisms often lead to much smaller backdoor sets. We show experimentally that instances from real-world domains that have thousands of variables often have backdoors of only a few variables. Our results suggest that structural notions explored for designing efficient algorithms for combinatorial problems should capture both statically and dynamically identifiable properties. We also evaluate the effect of different preprocessors on the structure of the instances and hence on the backdoor size. Finally, we also look into strong backdoors for satisfiable instances and their relationship to solution counting.

Keywords Boolean satisfiability (SAT) · problem structure · backdoor sets · dynamic simplification

1 Introduction

General purpose inference engines for Constraint Satisfaction Problems (CSPs) have witnessed tremendous progress since the early 1990s, and are increasingly becoming key components of efficient solution methodologies for many interesting problems

B. Dilkina · C.P. Gomes · A. Sabharwal
Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, U.S.A.
E-mail: {bistra,gomes,sabhar}@cs.cornell.edu

within the realm of artificial intelligence as well as hardware and software verification. While the initial work in this area was directed towards solving randomly generated problem instances, 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. Such instances encode highly combinatorial problems finding a solution to which in a brute-force manner is impractical. These problems tend to be NP-hard, leaving little room for an algorithmic technique that is guaranteed to be efficient at solving all instances of the underlying problem. Fortunately, while being able to efficiently solve all instances is highly unlikely, many real-world instances have been found to exhibit a "structure" very different from that of hard random instances, a structure that 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. The focus of this work is the study of such structure and, specifically, the tradeoff between the strength and the simplicity of various notions of structure.

Capturing and exploiting problem structure is key to solving large real-world combinatorial problems. For example, several interesting tractable classes of combinatorial problems have been identified by restricting the constraint language used to characterize such problem instances. Well-known cases include 2CNF, Horn, Linear Programming (LP), and Minimum Cost Flow problems (MCF). In general, however, such restricted languages are not rich enough to characterize complex 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 from the topology 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 — it is known that an instance is solvable in polynomial time if the treewidth of its constraint graph is bounded by a constant [7, 8, 14–16, 31]. Interestingly, even though the notion of bounded treewidth is defined with respect to tree decompositions, it is also possible to design algorithms for constraint satisfaction problems of bounded (generalized) hypertree width that do not perform any form of tree decomposition (see e.g., [5]). Other useful structural properties consider the nature of the constraints, such as their so-called functionality, monotonicity, and row convexity [9, 34].

Another approach for studying combinatorial problems focuses on the role of *hidden structure* as a way of analyzing and understanding the efficient performance of state-of-the-art constraint solvers on many real-world problem instances. One example of such hidden structure is a backdoor set, i.e., a set of variables such that once they are instantiated, the remaining problem *simplifies* to a tractable class [6, 19, 24, 30, 33, 35, 36]. Note that the notion of tractability in the definition of backdoor sets is not necessarily syntactically defined: it may often be defined only by means of a polynomial time algorithm, such as unit propagation. In fact, the notion of backdoor

sets came about as a way of explaining the high variance in performance of state-of-the-art Boolean Satisfiability (SAT) solvers, in particular heavy-tailed behavior, and as a tool for analyzing and understanding the efficient performance of these solvers on many real-world instances, in which the propagation mechanisms of fast "sub-solvers" play a key role. In this work the emphasis was not so much on efficiently identifying backdoor sets, but rather on the fact that many real-world instances have surprisingly small sets of backdoor variables and that once a SAT solver instantiates these variables, the rest of the problem is solved easily. In this context, randomization and restarts play an important role in searching for small backdoor sets [35, 36].

Even though variable selection heuristics, randomization, and learning in current SAT/CSP solvers are quite effective at finding relatively small backdoors in practice, finding a *smallest backdoor* is in general intractable in the worst case. This intractability result assumes that the size of the smallest backdoor is unknown and can grow arbitrarily with n. However, if the size of the backdoor is small and fixed to k, one can search for the backdoor 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 fixed k, although for moderate values of k the run time becomes infeasible in practice. Can one do better? This is a question considered in the area of fixed-parameter complexity theory. A problem with input size n and a parameter k is called *fixed-parameter tractable* w.r.t. k if it can be solved in time $O(f(k)n^c)$ where f is any computable function and c is a constant. Note that c does not depend on k, meaning that one can in principle search fairly efficiently for potentially large backdoors if backdoor detection for some class is shown to be fixed parameter tractable. Indeed, Nishimura, Ragde, and Szeider [28] showed that detecting strong backdoors (cf. Section 2 for a formal definition) w.r.t. the classes 2CNF and Horn is NP-complete but, interestingly, fixed-parameter tractable. This result for 2CNF and Horn formulas exploits the equivalence between (standard) strong backdoors and "deletion" backdoors, i.e., a set of variables that once deleted from a given formula (without simplification) make the remaining formula tractable. Note, however, that this result is only w.r.t. the tractable classes of *pure* 2CNF/Horn. In particular, certain kinds of obvious inconsistencies are not detected in these classes, such as having an empty clause in an *arbitrary* formula — clearly, any basic solver detects such inconsistencies. We show that such a seemingly small feature increases the worst-case complexity of backdoor identification, but, perhaps more importantly, can dramatically reduce the size of the backdoor sets.

More specifically, we prove that strong Horn- and 2CNF-backdoor identification becomes both NP- and coNP-hard, and therefore strictly harder than NP assuming NP $\neq$ coNP, as soon as empty clause detection is added to these classes. This increase in formal complexity has however also a clear positive aspect in that adding empty clause detection often considerably reduces the backdoor size. For example, we found that in certain graph coloring instances with planted cliques of size 4, while strong Horn-backdoors involve $\approx 67\%$ 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.

Encouraged by the positive effect of slightly extending our notion of Horn-backdoor with empty clause detection, we also considered backdoors w.r.t. RHorn

(renamable Horn), UP (unit propagation), PL (pure literal rule), UP+PL, PROB, PROBPL and SATZ. For each of these notions, we show on a variety of domains that the corresponding backdoors are significantly smaller than pure, strong Horn-backdoors. For example, we consider the smallest deletion RHorn-backdoors. We provide a 0-1 integer programming formulation for finding such optimal backdoors, and show experimentally that they are in general smaller than strong Horn-backdoors. In the three synthetic domains that we evaluated, deletion RHorn-backdoors are slightly smaller that strong Horn-backdoors. More interestingly, when considering real-world instances the differences is considerable. For a car configuration problem, while strong Horn-backdoor sets vary in size between 10-25% of the variables, deletion RHorn-backdoor sets vary only between 3-8%.

At a higher level, *our results show that the size of backdoors can vary dramatically depending on the effectiveness of the underlying simplification and propagation mechanism.* For example, as mentioned earlier, empty clause detection can have a major impact on backdoor size. Similarly, Horn versus RHorn has an impact. We also show that there can be a substantial difference between deletion backdoors, where one simply removes variables from the formula, versus strong backdoors, where one factors in the variable settings and considers the propagation effect of these settings. Specifically, we contrast deletion RHorn-backdoors with strong RHorn-backdoors. We prove by construction that there are formulas for which deletion RHorn-backdoors are exponentially larger than the smallest strong RHorn-backdoors.

Despite the worst-case complexity results for strong backdoor detection, we show that *Satz-Rand* [20, 25] is remarkably good at finding small strong backdoors on a range of experimental domains. For example, in the case of our graph coloring instances, the fraction of variables in a small strong SATZ-backdoor converges to zero as the size of the graph grows. For the car configuration problem, strong SATZ-backdoor sets involve 0-0.7% of the variables. We next consider synthetic logistics planning instances over n variables that are known to have strong UP-backdoors of size $\log n$ [21]. For all these instances, the size of the strong SATZ-backdoor sets is either zero or one. In contrast, the size of deletion RHorn-backdoors corresponds to over 48% of the variables, increasing with n. We also consider instances from game theory for which one is interested in determining whether there is a pure Nash equilibrium. For these instances, while strong Horn-backdoors and deletion RHorn-backdoors involve $\approx$ 68% and $\approx$ 67% of the variables, respectively, strong SATZ-backdoors are surprising small at less than 0.05% of the variables.

In addition to on-the-fly simplification techniques used by current solvers such as unit propagation and pure literal rule, another family of still poly-time but more sophisticated algorithms are so-called *preprocessing techniques.* Such preprocessors incorporate simplification and inference mechanism that are too expensive to be applied at each node of the backtrack search tree. Instead, they are applied only at the root of the search tree. The implied assumption is that the formulas obtained after running a preprocessor are easier to decide. However, preprocessing inevitably affects the structure of the original instance and hence may obscure some of the original hidden structure. Here, we look at the effect of preprocessing on the backdoor size in unsatisfiable SAT instances. Our results suggest that while preprocessing can some-

times reduce the backdoor size, it can also increase the backdoor size by obfuscating structure w.r.t. the particular sub-solver under consideration.

While the notion of strong backdoors is defined with respect to both satisfiable and unsatisfiable instances, most reported results to date study strong backdoors for unsatisfiable instances [24]. In this work, we look at strong backdoor sizes in some *satisfiable instances* as well, and the effect of preprocessors on the backdoors of these formulas. Finally, We explore the semantics of strong backdoors in satisfiable instances and their relation to *counting solutions*. In particular, while pure literal backdoors only facilitate the decision of satisfiability, strong backdoors w.r.t. unit propagation are also backdoors for the problem of counting solutions.

In summary, our results show that real-world SAT solvers such as *Satz* 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 properties, such as being Horn, but also properties that capture key aspects of state-of-the-art constraint solvers, such as unit propagation and pure literal rule.

The remainder of the article is organized as follows. Section 2 formally introduces the basic concepts of backdoor sets and commonly employed sub-solvers such as unit propagation and pure literal elimination. It also discusses empty clauses detection and deletion backdoors, and provides a summary of related work. Section 3 presents the two 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. The rest of the article presents an empirical study of backdoor sets. Section 4 discusses methods to compute (or approximate) the smallest backdoors for our study, and involves translation of the given problem to an integer programming problem. Section 5 discusses experimental results for the smallest backdoor size w.r.t. various static and dynamic sub-solvers. Section 6 explores various other directions, namely, the effect of preprocessing on the size of the smallest backdoor set, strong backdoor sets for satisfiable instances, and backdoors for the model counting problem. Finally, Section 7 concludes with a summary of the work.

2 Background and Related Work

We begin with an introduction to the Boolean satisfiability problem and CNF formulas, and then describe the notion of backdoor sets.

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}, $\varepsilon \in \{0,1\}$. $var(F)$ denotes the variables occurring in F. A (partial) *truth assignment* (or assignment, for short) is a map $\tau : X_\tau \to \{0,1\}$ defined on some subset of variables $X_\tau \subseteq var(F)$. A *solution* to a CNF formula F is a complete variable assignment τ (i.e., with $X_\tau = var(F)$) that satisfies all clauses of F. $F[\varepsilon/x]$ denotes the simplified formula obtained from F by removing all clauses that contain the literal x^{ε} and

removing, if present, the literal $x^{1-\varepsilon}$ from the remaining clauses. For a partial truth assignment τ, $F[\tau]$ denotes the simplified formula obtained by setting the variables according to τ.

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 $x^{1-\varepsilon}$ does not occur in F. A *Horn clause* is a clause that contains at most one positive literal. 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 two corresponding classes of formulas. *Renaming* or flipping a variable x in F means replacing every occurrence of x^{ε} in F with $x^{1-\varepsilon}$. F is *Renamable Horn*, also called RHorn, if all clauses of F can be made Horn by flipping a subset of the variables. Following Nishimura et al. [28], we define the *deletion* of a variable x from a formula F as syntactically removing the literals of x from 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.

The concept of backdoors and their theoretical foundations were introduced by Williams, Gomes, and Selman [35, 36]. Informally, a strong backdoor set is a set of variables such that for each possible truth assignment to these variables, the simplified formula is tractable. The notion of tractability is quite general, and it even includes tractable classes for which there is not a clean syntactic characterization. It is formalized in terms of a polynomial time sub-solver:

Definition 1 (sub-solver [35]) A *sub-solver* S is an algorithm that given as input a formula F satisfies the following conditions:

1. *Trichotomy:* S either rejects F or correctly determines it (as unsatisfiable or satisfiable, returning a solution if satisfiable),
2. *Efficiency:* S runs in polynomial time,
3. *Trivial solvability:* S can determine if F is trivially true (has no clauses) or trivially false (has an empty clause, $\{\}$), and
4. *Self-reducibility:* If S determines F, then for any variable x and value $\varepsilon \in \{0,1\}$, S determines $F[\varepsilon/x]$.

Definition 2 (strong S-backdoor [35]) A set B of variables is a *strong backdoor set* for a formula F w.r.t a sub-solver S if $B \subseteq var(F)$ and for every truth assignment $\tau : B \rightarrow \{0,1\}$, S returns a satisfying assignment for $F[\tau]$ or concludes that $F[\tau]$ is unsatisfiable.

Clearly, if B is a strong S-backdoor for F, then so is any B' such that $B \subseteq B' \subseteq var(F)$. 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}: we can formulate it as, "does *there exist* a $B \subseteq var(F), |B| = k$, such that *for every* truth assignment $\tau : B \rightarrow \{0,1\}$, S correctly determines $F[\tau/B]$?" We are interested in the complexity of this problem for specific sub-solvers.

The most *trivial sub-solver* that fulfills the conditions in Definition 1 is the one that only checks for the empty formula and for the empty clause. Lynce and Marques-Silva [26] show that the search effort required by the SAT solver *zChaff* [27] to

prove a random 3-SAT formula unsatisfiable is correlated with the size of the strong backdoors w.r.t. this trivial sub-solver.

More relevant sub-solvers employed by most state-of-the-art SAT solvers are Unit Propagation and Pure Literal Elimination, and their combination. 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 yes, it assigns the variable in the unit clause the corresponding satisfying value, and recurses on the simplified formula. If the formula does not contain any more unit clauses, it is rejected. The Pure Literal Elimination sub-solver (PL) checks for variables that appear as pure literals, assigning them the corresponding value and simplifying, until the formula is trivially solvable or is rejected (when no more pure literals are found). The sub-solver that uses both of these rules is referred to as UP+PL.

We note that unit propagation by itself is known to be sufficient for computing a satisfying assignment for any satisfiable Horn formula: set variables following unit propagation until there are no more unit clauses, and set the remaining variables to 0. A similar result is known for RHorn formulas. Interestingly, this does not mean that the smallest UP-backdoors are never larger than Horn- and RHorn-backdoors. For example, any (satisfiable) Horn formula with $k \geq 2$ literals per clause has a strong Horn-backdoor of size zero but no strong UP-backdoor of size $k-2$.

Szeider [33] studied the complexity of finding strong backdoors w.r.t. the above sub-solvers. For $S \in \{\text{UP, PL, UP+PL}\}$ and with k as the parameter of interest, he proved that the problem of deciding whether there exists a strong C-backdoor of size k is complete for the parameterized complexity class W[P]. Interestingly, the naïve brute-force procedure for this problem is already in W[P]; it 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. Hence, the in the worst case we cannot hope to find a smallest strong backdoor w.r.t. UP, PL, or UP+PL more efficiently than with brute-force search.

Satz [25] is a DPLL-based SAT solver that incorporates a strong variable selection heuristic and an efficient simplification strategy based on UP and PL. Its simplification process and lookahead techniques can be thought of as a very powerful sub-solver. Kilby et al. [24] study strong SATZ-backdoors: sets of variables such that for every assignment to these variables, *Satz* solves the simplified formula without any branching decisions (i.e., with a "branch-free" search). They 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.

The lookahead technique employed by *Satz* is a limited form of *probing*. Probing, also known as the *failed literal rule*, consists of assigning a variable to a value and running unit propagation. If a contradiction is found, then the variable is safely assigned to the opposite value. Otherwise, we test the opposite value and if contradiction is found, then we safely assign the variable to the first value attempted. Probing can also be thought of as a poly-time sub-solver. The sub-solver checks for variables that can be assigned by probing, assigning them the corresponding value and simplifying, until the formula is trivially solvable or is rejected (when no more variables can be assigned by probing). This sub-solver will need to apply UP at most n^2 times

and hence still runs in polynomial time. We refer to backdoors w.r.t. this sub-solver as PROB-backdoors.

A sub-solver S correctly determines a subclass of CNF formulas and rejects others, and hence implicitly defines the class C_S of formulas that it can determine. A natural variation of the definition of the backdoor does not explicitly appeal to a sub-solver, but rather requires the remaining formula, after setting variables in the backdoor, to fall within a known tractable sub-class, such as 2CNF, Horn, or RHorn. We will refer to such backdoors as Horn-backdoor, RHorn-backdoor, etc. Note that this way of defining the backdoor de facto corresponds to relaxing the assumption of the sub-solver's trivial solvability and therefore trivially satisfiable or unsatisfiable formulas need not lie within the tractable class. For example, an arbitrary formula with an empty clause may not be Horn. Such formulas—with an empty clause in them—are important for our discussion and we use the following notation:

Definition 3 $C_{\{\}}$ is the class of all formulas that contain the empty clause, $\{\}$. For any class C of formulas, $C^{\{\}}$ denotes the class $C \cup C_{\{\}}$.

We will show that strong backdoors w.r.t. $\text{2CNF}^{\{\}}$ and $\text{Horn}^{\{\}}$ behave very differently, both in terms of the complexity of finding backdoors as well as backdoor size, compared to 2CNF and Horn. In our arguments, we will use two properties of formula classes defined next.

Definition 4 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.

Definition 5 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 of G_k have size larger than k.

Note that (pure) 2CNF, Horn, and RHorn are closed under removal of clauses, while $C^{\{\}}$ is in general not: removing the empty clause may put a formula outside $C^{\{\}}$. Further, 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 set of k variables of G_k leaves a clause with three positive literals, which is neither 2CNF nor Horn.

A different notion of backdoors, motivated by the work of Nishimura et al. [28], 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). Formally,

Definition 6 (deletion C-backdoor [28]) A set B of variables is a deletion backdoor set of a formula F w.r.t. a class C if $B \subseteq var(F)$ and $F - B \in C$.

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 trivially in NP. This problem is in fact NP-complete when C is 2CNF [28], Horn [28], or RHorn [4].

In general, a deletion C-backdoor may not be a strong C-backdoor. E.g., when C includes $C_{\{\}}$, any 3CNF formula F has a trivial deletion C-backdoor of size 3: select any clause and use its variables as the deletion backdoor. Unfortunately, such a

"backdoor" set is of limited practical use for efficiently solving F. When the class C is closed under removal of clauses, every deletion C-backdoor is indeed also a strong C-backdoor. Conversely, strong C-backdoors often are not deletion C-backdoors, because assigning values to variables usually leads to further simplification of the formula. Nonetheless, for $C \in \{2\text{CNF, Horn}\}$, deletion and strong backdoors are equivalent, a key fact underlying the fixed parameter algorithm of Nishimura et al. [28]. We will show that this equivalence between deletion backdoors and strong backdoors does not hold for RHorn.

Paris et al. [30] studied deletion RHorn-backdoors. They proposed a two step approach: find a renaming that maximizes the number of Horn clauses using a local search method and then greedily delete variables from the remaining non-Horn clauses until the renamed formula becomes Horn. The variables deleted in the second step form a deletion RHorn-backdoor. They find that branching on these variables can significantly speed up DPLL solvers.

3 Theoretical Results

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 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. 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

We show that strong backdoors w.r.t. $2\text{CNF}^{\{\}}$ and $\text{Horn}^{\{\}}$ behave very differently, both in terms of the complexity of finding backdoors as well as backdoor size, compared to strong backdoors w.r.t. 2CNF and Horn.[1] In particular, we prove that the two problems of deciding whether a formula has a strong backdoor w.r.t. $2\text{CNF}^{\{\}}$ and $\text{Horn}^{\{\}}$, respectively, are NP-hard as well as coNP-hard. This shows that unless NP=coNP, this problem is much harder than detecting strong backdoors w.r.t. 2CNF and Horn, which are both known to be NP-complete, and thus within NP [28].

We start with a relatively simple observation, which highlights the potential positive impact of adding empty clause detection. Consider the unsatisfiable formula $F = (x_0) \wedge (\neg x_0) \wedge (x_0 \vee x_1 \vee \ldots \vee x_n)$. It is clear that the variable x_0 is the "cause" of

[1] Recall that adding $C_{\{\}}$ to 2CNF and Horn corresponds to adding empty clause detection to the sub-solvers corresponding to the two classes.

F being unsatisfiable. Indeed, $\{x_0\}$ is a strong $C^{\{\}}$-backdoor of size 1 for F. On the other hand, one must assign values to at least n (or $n-1$) variables before the third clause of F becomes Horn (or 2CNF, resp.). In other words, this examples shows that empty clause detection can lead to not only exponentially but even arbitrarily smaller backdoor sets.

Proposition 1 *There are formulas for which there is a strong $C^{\{\}}$-backdoor of size 1 but all strong Horn- or 2CNF-backdoors involve nearly all the variables.*

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. [28] for (pure) 2CNF/Horn using a reduction from the NP-complete Vertex Cover problem. In the other direction, we prove coNP-hardness of backdoor detection w.r.t. Horn$^{\{\}}$ and 2CNF$^{\{\}}$ 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 Horn and 2CNF are closed under clause deletion. These two results are formally stated as Lemmas 1 and 2 below, which together lead to our main result, Theorem 1.

Lemma 1 *Let $C \in \{$Horn, 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. [28] for (pure) 2CNF/Horn. 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? Recall that a vertex cover U is a subset of V such that every edge in E has at least one end point in U. Given an instance $\langle G,k\rangle$ of this problem, we will construct a formula F_{Horn} with all positive literals such that F_{Horn} has a strong Horn$^{\{\}}$-backdoor of size k iff G has a vertex cover of size k. Similarly, we will construct F_{2CNF}.

F_{Horn} has $|V|$ variables and $|E|$ clauses. The variables are x_v for each $v \in V$. For each edge $e = \{u,v\} \in E, u < v$, F_{Horn} contains the binary clause $(x_u \vee x_v)$. It is easy to see that if G has a vertex cover U, then the corresponding variable set $X_U = \{x_u \mid u \in U\}$ is a strong Horn$^{\{\}}$-backdoor: for any assignment τ to X_U, $F[\tau/X_U]$ only contains unit clauses or the empty clause, and is thus in Horn$^{\{\}}$. For the other direction, suppose X_U is a strong Horn$^{\{\}}$-backdoor. We claim that variables of X_U must touch every clause of F_{Horn} so that the corresponding vertices U touch every edge of G and thus form a vertex cover. 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.

F_{2CNF} has $|V|+|E|$ variables and $|E|$ clauses. The variables are x_v for each $v \in V$ and $y_{u,v}$ for each $\{u,v\} \in E, u < v$. For each such $\{u,v\}$, F_{2CNF} contains the ternary clause $(x_u \vee x_v \vee y_{u,v})$. The argument for the correctness of the reduction is very similar to above, 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, so that there is no direct way to obtain 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 with fewer (and eventually no) such y variables. □

We now prove coNP-hardness of backdoor detection w.r.t. Horn$^{\{\}}$ and 2CNF$^{\{\}}$, exploiting the notions introduced in Definitions 4 and 5.

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 iff 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 iff 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 $G[\tau_G/B_G] \notin C$. Because of the closure of C under removal of clauses and 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 $F[\tau/B] \in C_{\{\}}$. Since $G[\tau_G/B_G] \notin C_{\{\}}$, it must be that $H[\tau_H/B_H] \in C_{\{\}}$ 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 main theorem:

Theorem 1 *Let $C \in \{$Horn, 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, and thus harder than both NP and coNP, assuming NP $\neq$ 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 RHorn backdoors are more likely to succinctly capture structural properties of interest in formulas than 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 Let $F = (x_1 \vee x_2) \wedge (\neg y_1 \vee \neg y_2) \wedge (\neg x_1 \vee y_1 \vee z) \wedge (\neg x_1 \vee y_2 \vee \neg z) \wedge (\neg x_2 \vee y_1 \vee z) \wedge (\neg x_2 \vee y_2 \vee \neg z)$. First we observe that $B = \{z\}$ is a strong RHorn-backdoor for F because for $z = 0$ we can rename x_1 and y_1, and for $z = 1$ we can rename x_1 and y_2 to get a Horn formula. On the other hand, $\{z\}$ certainly does not work as a deletion backdoor because we must rename at least one of x_1 and x_2, which forces both y_1 and y_2 to be renamed and violates the Horn property. In fact, it can be easily verified that both $\{x_1\}$ and $\{y_1\}$ are also not 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: $\{x_i \mid 1 \leq i \leq t\}$, $\{y_j \mid 1 \leq j \leq s\}$, and $\{z_k \mid 1 \leq k \leq \log_2 s\}$. 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 iff 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:

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

We now analyze RHorn-backdoors for F_n. First, we show that $\{z_k \mid 1 \leq k \leq \log_2 s\}$ 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. □

4 Experimental Setup

The remainder of this article presents an empirical study of the size of backdoor sets in various settings w.r.t. a number of static and dynamic sub-solvers. The present section describes our methodology (i.e., how we compute or approximate the smallest backdoor sets) and the benchmark domains used for experiments. We will then present are main results in Section 5, followed by some additional extensions in Section 6.

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. the solver *Satz*. We also consider computing bounds on sizes of smallest backdoors w.r.t. unit propagation, pure literal elimination, and their combination.

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 [28]. 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 a 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) 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 not-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:

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

The first set of constraints ensures that each Boolean variable x_i is either not-renamed, renamed, or deleted. The second set of constraints ensures that each clause is Horn (in each clause, the total number of not-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 ILOG CPLEX libraries [23] for experimenting with Horn- and RHorn-backdoors.

Smallest Strong SATZ-, PROBPL-, PROB-, (UP+PL)-, UP-, and PL-Backdoors: Following previous work [24, 35], we also use the complete randomized solver *Satz-Rand* [20] to find small strong backdoors. *Satz-Rand* employs a randomized variable selection heuristic as well as a laborious propagation procedure at each search node that includes limited probing, unit propagation and pure literal elimination. The limited probing consists of a single pass through the variables. If it finds a variable that it can set by probing it does and applies UP, but once it has iterated once over the variables it then moves to heuristically selecting a branch variable. We refer to the backdoors with respect to the this built-in propagation as SATZ-backdoors. We obtain an upper bound on the size of the smallest strong SATZ-backdoor by running *Satz-Rand*

(a randomized version of *Satz*) without restarts multiple times with different seeds and recording the set of variables on which the solver branches when proving unsatisfiability. For every possible assignment to this set of variables, *Satz* decided the simplified formula in a branch-free manner by applying its propagation mechanism, and the size of this variable set is an upper bound on the minimum SATZ-backdoor size.

In our initial experiments [11], we recorded the set of variables not set by UP and PL while searching with *Satz-Rand* to obtain UPPL-backdoors, i.e. the UPPL-backdoor includes the variables set by probing plus the variables set by branching. Similarly, the set of variables not set by UP while searching with *Satz-Rand*, i.e. the variables set by probing, PL or branching, is a UP-backdoor. Recording backdoors in such a way, conditioned the backdoors with respect to UPPL and PL to include the variables set by probing and PL which are by default applied by *Satz-Rand*. More recently, we have instrumented the *Satz-Rand* code so that we can turn on/off each of the propagation techniques (probing, UP and PL) while keeping the randomized branch heuristic, thereby removing this conditioning. In addition, probing is applied until closure similarly to UP and PL, i.e. until no variable can be determined by probing – as opposed to the default where *Satz* scans the variables with probing only once. We run the instrumented *Satz-Rand* with probing (which includes UP) as propagation procedure (but turning off PL) and record the branch variables. By a similar reasoning as above, the size of this set gives us an upper bound on the smallest strong PROB-backdoor size. We combine probing and PL, to obtain backdoors w.r.t. PROBPL. Similarly, we record all branch variables in *Satz-Rand* using only UP and PL (or UP only or PL only) propagation procedures to obtain an upper bound on the smallest strong UPPL (or UP or PL)-backdoor size. For satisfiable instances, we obtain upper bounds on the smallest backdoors by forcing *Satz-Rand* to continue searching even after a solution is found until the full search space is explored. By recording the variables on which it branches while finding all solutions, we obtain a strong backdoor set such that for each assignment the simplified formula is determined as either unsatisfiable or satisfiable by the propagation mechanism of *Satz*. For each problem instance, we record the smallest backdoor size found across all runs as the upper bound to the minimum strong backdoor size for this instance.

4.2 Benchmark Domains

We consider 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 are generated using the clique hiding graph generator of Brockington and Culberson [3]. All instances were generated with the probability of adding an edge equal to 0.5 and with a hidden clique of size 4. All SAT-encoded instances are unsatisfiable when the number of colors is 3. The twelve variables representing color assignments to the four vertices in the hidden 4-clique constitute a strong $C_{\{\}}$-backdoor, since any assignment of colors to these four vertices will fail at least one coloring constraint.

MAP planning: This is a synthetic logistics planning domain for which the size of strong UP-backdoors is well understood [21]. 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, encoding one planning step less than the length of the optimal plan. Hoffmann et al. [21] identify that MAP instances with $k = 2n-3$ (called asymmetric) have logarithmic size DPLL refutations (and backdoors). We evaluate the size of the backdoors in asymmetric MAP instances of various sizes ($n = 5..50$).

Game theory: These instances encode the problem of computing an equilibrium strategy in a multi-player game and were used first in a study of the existence of such equilibria under varying network topology [10]. In a game, interactions between players can be represented by an undirected graph where nodes represent players and edges represent mutual dependencies between players. 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. Here we consider binary games, where each player has exactly two action choices. Our focus will be 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 $\mathbb{G}(n,p)$. In a pure Nash equilibrium (PNE), 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 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.

Car configuration: This is a real-world SAT benchmark whose instances encode problems from the validation and verification of automotive product configuration data for the Daimler Chrysler's Mercedes car lines [32]. We consider a set of unsatisfiable instances available at `http://www-sr.informatik.uni-tuebingen.de/~sinz/DC/`.

Bridge Faults: This benchmark set is part of SATLIB [22] and is listed as BF. This is a real-world SAT benchmark whose instances encode problems from the verification of bridge faults. The instances are generated by Nemesis, a test-pattern generation program for realistic bridging faults in CMOS Integrated Circuits. We also give results on the instance ssa0432-003, which is one of the eight instances which test for single-at-stuck faults (also part of the BF benchmark set).

5 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 efficient performance of state-of-the-art solvers on many real-world instances. Previous work of strong backdoors has considered random SAT formulas [24, 26]. In this section, we demonstrate that,

Table 1 Strong backdoor sizes as a percentage of the number of variables for various ensembles of instances: Graph Coloring (gcp), MAP planning (map), Pure Nash Equilibrium (pne), Automotive Configuration (cnnn), and Bridge Faults (ssa/bf). Each row reports the average over several instances for gcp, pne and cnnn. Backdoor sizes are shown as the average % of the number of problem variables. The RHorn numbers are for deletion backdoors. Horn- and RHorn-backdoor sizes are for the smallest corresponding backdoors (except the ones indicated with *), while the rest are upper bounds.

instance	num	num	Static (optimal, %)		Dynamic (upper bound, %)						
set	vars	clauses	Horn	RHorn	PROBPL	(SATZ)	PROB	UP+PL	UP	PL	$C_{\{\}}$
Graph Coloring											
gcp_100	300	7,557.7	66.67	*57.73	0.33	(0.33)	0.33	1.43	1.47	34.63	4.00
gcp_200	600	30,122.0	66.67	*62.47	0.17	(0.17)	0.17	0.73	0.73	30.43	2.00
gcp_300	900	67,724.4	66.67	*63.82	0.11	(0.11)	0.11	0.51	0.51	27.36	1.33
gcp_400	1,200	119,997.4	66.67	*64.48	0.08	(0.08)	0.08	0.38	0.40	26.82	1.00
gcp_500	1,500	187,556.0	66.67	*64.88	0.07	(0.07)	0.07	0.40	0.41	32.75	0.80
AI Planning											
map_5_7	249	720	38.96	37.75	0	(0)	0	2.41	2.41	62.65	
map_10_17	1,284	5,000	44.55	44.31	0	(0)	0	1.25	1.25	59.27	
map_20_37	5,754	33,360	47.31	47.25	0	(0)	0	0.63	0.63	57.72	
map_30_57	13,424	103,120	48.21	48.19	0	(0)	0	0.42	0.42	57.22	
map_40_77	24,294	232,280	48.66	48.65	0	(0)	0	0.31	0.31	56.97	
map_50_97	38,364	438,840	48.93	48.92	0	(0)	0	0.25	0.25	56.83	
Game Theory											
pne	2,000	40,958.9	67.88	66.86	0.00	(0.05)	0.00	0.07	0.09	0.22	
pne	3,000	60,039.7	67.66	66.55	0.00	(0.00)	0.00	0.03	0.04	0.08	
pne	4,000	78,839.1	67.97	66.93	0.00	(0.00)	0.00	0.03	0.03	0.07	
pne	5,000	98,930.8	67.81	66.80	0.00	(0.00)	0.00	0.02	0.02	0.05	
Automotive Configuration											
c168_fw_sz	1,698	5,646.8	14.32	2.83	0.06	(0.16)	0.06	0.45	0.60	4.16	
c168_fw_ut	1,909	7,489.3	23.62	5.50	0.00	(0.00)	0.00	0.06	0.06	10.84	
c170_fr_sz	1,659	4,989.8	9.98	3.57	0.00	(0.13)	0.00	0.45	0.48	3.40	
c202_fs_sz	1,750	6,227.8	12.31	4.55	0.00	(0.13)	0.00	0.21	0.27	4.11	
c202_fw_rz/sz	1,799	8,906.9	14.48	6.12	0.02	(0.22)	0.02	0.86	1.06	6.24	
c202_fw_ut	2,038	11,352.0	21.25	7.61	0.00	(0.00)	0.00	0.15	0.25	9.13	
c208_fA_rz/sz	1,608	5,286.2	10.52	4.51	0.00	(0.06)	0.00	0.32	0.37	2.30	
c208_fA_ut	1,876	7,335.5	23.13	7.46	0.00	(0.00)	0.00	0.08	0.08	13.33	
c208_fc_rz	1,654	5,567.0	10.28	4.59	0.00	(0.36)	0.00	0.67	0.79	3.20	
c208_fc_sz	1,654	5,571.8	10.47	4.68	0.01	(0.16)	0.01	0.40	0.76	3.17	
c210_fs_rz	1,755	5,764.3	11.64	4.22	0.00	(0.55)	0.00	0.55	0.68	3.30	
c210_fs_sz	1,755	5,796.8	11.77	4.35	0.00	(0.30)	0.00	0.39	0.54	3.51	
c210_fw_rz	1,789	7,408.3	12.54	4.81	0.00	(0.65)	0.00	0.65	0.86	4.14	
c210_fw_rz/sz	1,789	7,511.8	13.74	5.37	0.02	(0.23)	0.02	0.41	0.73	5.62	
c210_fw_ut	2,024	9,720.0	20.73	7.31	0.00	(0.00)	0.00	0.37	0.42	12.80	
c220_fv_rz/sz	1,728	4,758.2	9.14	2.92	0.09	(0.19)	0.16	0.33	0.54	5.53	
Verification											
bf0432-007	1,040	3,668	50.10	*28.17	1.54		1.54	11.54	12.69	85.19	
bf1355-075	2,180	6,778	52.06	*26.15	2.80		2.80	5.00	5.00	64.17	
bf1355-638	2,177	6,768	51.68	*25.86	2.07		2.07	7.95	8.04	62.93	
bf2670-001	1,393	3,434	41.92	*20.96	1.22		1.22	1.29	1.65	48.96	
ssa0432-003	435	1,027	48.97	20.46	0.00		0.00	3.91	3.91	88.51	

although one cannot always efficiently identify the smallest backdoor sets, in practice many real-world combinatorial problems have surprisingly small backdoors.

Specifically, in our experimental evaluation, we compute the size of strong and deletion backdoors w.r.t. several classes and sub-solvers in five problem domains, discussed in Section 4. The results are shown in Table 1. The results for SATZ-backdoors are presented in braces in the PROBPL column, since SATZ propagation is a limited form of PROBPL.

The MIP problems, encoding finding minimum deletion RHorn-backdoors for the graph coloring and the bridge fault instances, are actually very hard to solve. Here we report results based on the best feasible solutions found in a limited computational

time, i.e. an upper bound on the minimum deletion RHorn backdoor sizes. For all other instances we report optimal del RHorn-backdoor sizes.

The graph coloring domain illustrates that both 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 a $C_{\{\}}$-backdoor of size 12. We note that *Satz* finds backdoors even smaller than the $C_{\{\}}$-backdoor.

In the MAP planning domain, strong Horn-backdoors and deletion RHorn-backdoors are of comparable size and relatively large (37-48%); as expected strong UP-backdoors are quite small. Interestingly, *Satz* solves these instances without any search at all when using its full propagation procedure. The smallest strong PROBPL- and PROB-backdoors are of size 0.

For the game theory problems of computing a pure nash equilibrium, while strong Horn-backdoor sets and deletion RHorn-backdoor involve $\approx 68\%$ and $\approx 67\%$ of the variables, respectively, for the selected instances strong PROB-backdoors are surprisingly small, close to 0% of the variables.

In the automotive product configuration problems, while strong Horn-backdoors vary between 10-25% of the variables, RHorn-backdoor sets are considerably smaller at 3-8%. Notice that the RHorn-backdoor sizes are similar to the PL-backdoor sizes. However, these are deletion RHorn backdoor sizes and hence provide only an upper bound on the minimum strong RHorn-backdoor size, which could be even smaller. Strong PROBPL-backdoors involve only 0-0.09% of the variables. In this domain, the full probing included in the PROB- and PROBPL- sub-solvers results in finding smaller strong backdoors than the SATZ-backdoors where the propagation only includes limited probing (with backdoor sizes of 0-0.7% instead).

Finally, in the bridge fault instances, we can again see that deletion RHorn-backdoor sets are considerably smaller than Horn-backdoor sets, involving 20-29% versus 41-51% of the variables. Pure literal elmination is not effective on these problems, resulting in PL-backdoors of 48-89% of the variables.

We can conclude that RHorn-backdoors are smaller than Horn-backdoors, especially in the industrial auto configuration and bridge fault benchmarks (as opposed to the three synthetic domains). Hence, between these two syntactic tractable classes, RHorn is computationally more relevant. Further, PL-backdoor sizes are larger than UP- and UPPL-backdoor sizes. In particular, adding PL propagation to UP to obtain UPPL, and adding PL to probing to obtain PROBPL propagation has minor effect on the size of the minimum strong backdoor. In fact, we observed that using PROBPL or UPPL as a sub-solver causes *Satz-Rand* to often find larger backdoors than when using PROB or UP respectively. In effect, the added pure literal propagation changes the structure of the problem in such a way that it misleads the solver heuristic toward less effective branch variables. This issue is further examined in the next section in the context of preprocessors and their effect on backdoor sizes. Because of this effect of pure literal elimination (which has been observed previously) and its computational overhead, PL has been omitted in some of the state-of-the-art solvers.

6 Backdoors: Preprocessing, Satisfiable Instances, and Model Counting

6.1 Effect of Preprocessing

In recent years, there has been interest in developing preprocessing techniques for SAT. Such preprocessing techniques change the structure of the original formula and hence might have an effect on the backdoor size. In this section, we investigate the effects that preprocessors have on the size of strong backdoors in unsatisfiable instances.

We consider four of the more popular preprocessors used with state-of-the-art SAT solvers: 3-Resolution, 2-SIMPLIFY, HyPre, and SatELite. The simplification techniques incorporated in these preprocessors are often more sophisticated but also more time-consuming than standard simplification techniques such as UP and PL, and hence are not cost-effective as a propagation mechanism at each search node. 3-Resolution is a preprocessing technique that has been used in several SAT solvers, in particular *Satz*. It resolves clauses of length at most 3 until saturation. 2-SIMPLIFY [2] efficiently implements and combines well-known 2-SAT techniques, a limited form of hyper-resolution and a novel use of transitive reduction to reduce formula size. HyPre [1] reasons on binary clauses similarly to 2-SIMPLIFY, but also incorporates full resolution. Also, unit propagation and equality reduction are applied until saturation. SatELite [13] uses the rule of Variable Elimination by Substitution.

When applied to structured formulas, as ones that appear in real-world domains, preprocessors often lead to great reduction in the size of the instance and sometimes can even solve the instance without search. In such cases, one can consider the preprocessor as a sub-solver for which the instance has a strong backdoor of size 0. In particular, the instances from the four domains studied in the previous section are often fully solved by the preprocessors themselves. For example, 3-Resolution determines the graph coloring problems, while the MAP domain is easy for HyPre and 3-Resolution.

Here, we report results on the BF domain from SATLIB [22], which consists of four benchmark instances which test for bridge faults. Table 2 reports the number of variables and clauses in the original formula and in the resulting formulas after applying each of the preprocessors. It also reports upper bounds on the minimum strong backdoor size w.r.t. SATZ and UP+PL. For some instances, certain preprocessors determine the unsatisfiability of the instance. In this case, we report 0 in all entries. As an example, the instance bf1355-075 is solved by SatELite, while it has a strong SATZ-backdoor of size 179 after preprocessing with 2-SIMPLIFY. The results in Table 2 show that none of the preprocessors has a monotonic effect on the backdoor size in unsatisfiable instances. SatELite overall has the most positive effect resulting in SATZ-backdoors of size 0 in three of the instances, however it results in larger backdoors for bf0432-007. On the other hand, 2-SIMPLIFY increases the backdoor size for three of the instances, but reduces the backdoor size to 8 for bf2670-001. The results seem to suggest that while for some instances preprocessing simplifies the formula, in some cases it obfuscates the hidden structure of the problem and results in larger backdoors.

Table 2 Effect of preprocessing. For each instance, we report upper bounds on the strong backdoors size w.r.t. SATZ and UP+PL for the original formula and the formulas obtained after running 2-SIMPLIFY, HyPre, SatELite, 3-Resolution.

instance name	preprocessor used	simplification		backdoor size (%)	
		num vars	num clauses	SATZ	UP+PL
bf0432-007.cnf	none	1,040	3,122	80	217
	2-SIMPLIFY	1,205	2,501	67	168
	HyPre	325	1,383	65	156
	SatELite	556	2,216	132	219
	3-Resolution	795	3,499	30	191
bf1355-075.cnf	none	2,180	5,146	44	117
	2-SIMPLIFY	2,583	4,134	179	221
	HyPre	704	3,124	82	103
	SatELite	0	0	0	0
	3-Resolution	1,682	5,557	72	158
bf1355-638.cnf	none	2,177	5,385	34	142
	2-SIMPLIFY	2,589	4,861	104	152
	HyPre	486	2,010	37	50
	SatELite	814	3,134	0	5
	3-Resolution	1,461	4,397	46	81
bf2670-001.cnf	none	1,393	2,926	16	32
	2-SIMPLIFY	1,592	1,629	8	29
	HyPre	0	0	0	0
	SatELite	80	294	0	10
	3-Resolution	1,202	2,970	7	19

6.2 Backdoors for Satisfiable Instances

Most of the previous work on strong backdoors has analyzed unsatisfiable instances [e.g. 26]. However, the notion of strong backdoor is also relevant to satisfiable instances. While for unsatisfiable instances, a strong backdoor set B is such that for every assignment to B, the sub-solver derives the empty clause when simplifying the formula. For satisfiable instances, a strong backdoor set B is such that for every assignment to B, the sub-solver either derives the empty clause (a subtree that contains no solutions), or it finds a solution that is an extension of the assignment to B. On the other hand, the notion of weak backdoors with respect to satisfiable formulas captures, in a sense, a "witness" to the satisfiability of the instance.

We study satisfiable instances with many solutions from the Car Configuration domain [32]. The results we obtain are very similar across instances, and we report on one representative instance. Table 3 suggests that for such satisfiable instances with many solutions, the weak backdoor size is extremely small, while the strong backdoor size is larger. A strong backdoor needs to capture a solution for each backdoor assignment that does not lead to an inconsistent formula. In addition, when considering strong backdoors, the choice of sub-solver has a major effect. Interestingly, although in general UP is a more effective propagation mechanism when deciding satisfiability, when considering strong backdoors in the satisfiable car configuration instances, the PL sub-solver can result in smaller backdoor sizes than those w.r.t. UP. Finally, with the exception of SatELite, preprocessors do not significantly affect the size of strong backdoors in satisfiable instances of the Car Configuration domain. Notice that when PL is added to UP, the strong backdoor sizes are dramatically reduced. This effect appears to be due to the fact that the pure literal rule is de facto a "stream-liner" [17, 18]: as PL assigns variables that appear as pure literals, it guaran-

Table 3 Weak and Strong backdoor sizes of a representative *satisfiable* Car Configuration instance of the original formula and the formulas obtained after running preprocessors 2-SIMPLIFY, HyPre, SatELite, 3-Resolution.

C168_FW_MT_28	vars	clauses	SATZ	UP+PL	UP	PL
Weak Original	1909	3808	18	22	423	357
Strong Original	1909	3808	153	156	500	388
Strong HyPre	536	2816	144	145	478	306
Strong SatELite	107	790	34	35	76	98
Strong 3-Resolution	228	1088	142	142	225	189
Strong 2-SIMPLIFY	1920	2618	118	118	478	285

tees that the satisfiability of the formula remains unchanged, while at the same time potentially pruning several solutions.

6.3 Backdoors for Model Counting

The observation about strong backdoors w.r.t. PL acting as a streamliner raises an interesting question about the relationship between strong backdoors and counting the number of solutions of the given formula (the *model counting* problem). Consider a strong backdoor B for a formula F w.r.t. a sub-solver S. One can count the number of solutions of F, denoted $\#F$, by adding up the solution counts $\#F[\tau/B]$ for each of the $2^{|B|}$ truth assignments τ to the variables in B. Suppose the sub-solver S has the property that there exists a poly-time algorithm S' such that whenever a formula G is determined by S as being satisfiable or unsatisfiable, then S' can compute $\#G$. If this property holds, then B also acts as a backdoor for the model counting problem: adding up $\#F[\tau/B]$ for all assignments τ to B yields a $2^{|B|}n^{O(1)}$ time algorithm for computing $\#F$.

Looking at various sub-solvers (and tractable classes) discussed so far, we may ask which ones have the above property, i.e., strong backdoors for which of these sub-solvers also act as backdoors for the model counting problem? Consider the pure literal sub-solver. A strong PL backdoor B may not necessarily help with model counting. For example, if for some assignment τ to B, the simplified formula $F[\tau/B]$ has *only* pure literals, the PL sub-solver sets all variables of $F[\tau/B]$ to their respective pure values (possibly eliminating several solutions in the process) and immediately declares the formula satisfiable. However, the model counting problem for formulas with only pure literals (sometimes called *monotone* formulas) is still #P-complete,[2] making it highly unlikely that a poly-time model counting algorithm for such formulas exists. Thus, a small strong PL backdoor does not necessarily yield an efficient way to compute $\#F$. Similarly, for 2CNF and Horn formulas, the corresponding model counting problem is known to be #P-complete, making strong backdoors for these classes (even with empty clause detection added) not very useful for model counting.

On the other hand, strong backdoors B w.r.t. certain common sub-solvers *do* yield a $2^{|B|}n^{O(1)}$ time algorithm for computing $\#F$. We present the case for the unit prop-

[2] #P-completeness for monotone 2CNF formulas can be proved by a simple reduction from model counting for the Vertex Cover problem.

agation sub-solver. Unlike PL, every variable assignment that UP makes when B is set to τ is a logical implication of the residual formula $F[\tau/B]$. Hence, setting those variables by UP does not affect the set of solutions of $F[\tau/B]$ as the UP sub-solver proceeds to either derive an empty clause or simplify $F[\tau/B]$ to the empty formula. In the former case, $F[\tau/B]$ clearly has zero solutions. In the latter case, we claim that $F[\tau/B]$ has exactly 2^m solutions, where m is the number of variables of F that are not in B and are also not assigned a value by UP. This is seen by noting that every variable of $F[\tau/B]$ set by UP must have been set that way in all solutions to $F[\tau/B]$, and every variable not set by UP is in fact a "don't care" variable for $F[\tau/B]$ and can be set either way in all solutions. This gives us a $2^{|B|}n^{O(1)}$ algorithm for computing $\#F$ as claimed.

In related work, another class of formulas for which counting is easy was considered by [29]. The class consists of "cluster formulas", which are variable disjoint union of so-called "hitting formulas", where any two clauses of a hitting formula "clash" in at least one literal. Again, given a backdoor B w.r.t. this class of formulas, counting the number of solutions of the original formula can be done in $2^{|B|}n^{O(1)}$ time. They also describe how to find such backdoors of bounded size (by relaxing to deletion backdoors which are not necessarily minimal strong backdoor) in formulas with bounded cluster width.

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 these problems. This work explores such structure, focusing in particular on the tradeoffs between static vs. dynamic properties 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 identifying a strong Horn- or 2CNF-backdoor (a static class) is known to be in NP and fixed parameter tractable, strong backdoor identification w.r.t. to Horn and 2CNF 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. While such a feature increases the worst-case complexity of finding backdoors, our experiments show that in practice it also has a clear positive impact: it reduces the size of the backdoors dramatically. For the class RHorn, we prove that deletion backdoors can be exponentially larger than strong backdoors, in contrast with the known results for 2CNF- and Horn-backdoors. Nonetheless, we show experimentally that deletion RHorn-backdoors can be substantially smaller than strong Horn-backdoors. We also demonstrate that strong backdoors w.r.t. UP, PL, and UP+PL 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.

Further, our experiments suggest that preprocessing does not have a consistent effect on the strong backdoors size for unsatisfiable formulas and can result in both

larger and smaller backdoors. On the other hand, preprocessing does not seem to affect in any significant way the backdoor size in satisfiable instances. In the context of strong backdoors w.r.t satisfiable instances and model counting, we also consider which sub-solvers lead to strong backdoors that also act as backdoors for the model counting problem. In particular, while PL cannot help with identifying backdoors for model counting, small strong UP-backdoors allow for efficiently counting solutions.

Acknowledgements 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 [11]. Part of this work was also presented at the (unarchived) 10th International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, FL in January 2008 [12]. 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).

References

1. F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Proceedings of SAT-03: 6th International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 341–355, Santa Margherita Ligure, Italy, May 2003.
2. R. I. Brafman. A simplifier for propositional formulas with many binary clauses. In *Proceedings of IJCAI-01: 17th International Joint Conference on Artificial Intelligence*, pages 515–522, Seattle, WA, Aug. 2001.
3. M. Brockington and J. C. Culberson. Camouflaging independent sets in quasi-random graphs. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring and Satisfiability: the Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 75–88. American Mathematical Society, 1996.
4. V. Chandru and J. N. Hooker. Detecting embedded Horn structure in propositional logic. *Information Processing Letters*, 42(2):109–111, 1992.
5. H. Chen and V. Dalmau. Beyond hypertree width: Decomposition methods without decompositions. In *CP-05: 11th International Conference on Principles and Practice of Constraint Programming*, pages 167–181, 2005.
6. H. Chen, C. P. Gomes, and B. Selman. 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 *Lecture Notes in Computer Science*, Paphos, Cyprus, Nov. 2001.
7. R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., 2003. ISBN 1558608907.
8. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987.
9. Y. Deville and P. Van Hentenryck. An efficient arc consistency algorithm for a class of CSP problems. In *Proceedings of IJCAI-91: 12th International Joint Conference on Artificial Intelligence*, pages 325–330, Sydney, Australia, Aug. 1991.
10. B. Dilkina, C. P. Gomes, and A. Sabharwal. The impact of network topology on pure Nash equilibria in graphical games. In *Proceedings of AAAI-07: 22nd Conference on Artificial Intelligence*, pages 42–49, Vancouver, BC, July 2007.
11. B. Dilkina, C. P. Gomes, and A. Sabharwal. Tradeoffs in the complexity of backdoor detection. In *CP-07: 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 256–270, Providence, RI, Sept. 2007.
12. B. Dilkina, C. P. Gomes, and A. Sabharwal. Tradeoffs in backdoors: Inconsistency detection, dynamic simplification, and preprocessing. In *ISAIM-08: 10th International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, FL, Jan. 2008.
13. N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of SAT-05: 8th International Conference on Theory and Applications of Satisfiability*

Testing, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75, St. Andrews, U.K., June 2005.
14. E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.
15. E. C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4): 755–761, 1985.
16. E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of AAAI-90: 8th National Conference on Artificial Intelligence*, pages 4–9, Boston, MA, 1990.
17. C. P. Gomes, A. Sabharwal, and B. Selman. Model counting: A new strategy for obtaining good bounds. In *Proceedings of AAAI-06: 21st Conference on Artificial Intelligence*, pages 54–61, Boston, MA, July 2006.
18. C. P. Gomes and M. Sellmann. Streamlined constraint reasoning. In *CP-04: 10th International Conference on Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 274–289, Toronto, Canada, Oct. 2004.
19. C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1-2):67–100, 2000.
20. C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of AAAI-98: 15th National Conference on Artificial Intelligence*, pages 431–437, Madison, WI, July 1998.
21. J. Hoffmann, C. Gomes, and B. Selman. Structure and problem hardness: Goal asymmetry and DPLL proofs in SAT-based planning. *Logical Methods in Computer Science*, 3(1-6), 2007.
22. H. H. Hoos and T. Stützle. SATLIB: An online resource for research on SAT. In I. P. Gent, H. van Maaren, and T. Walsh, editors, *SAT2000*, pages 283–292. IOS Press, 2000. URL `http://www.satlib.org`.
23. ILOG, SA. CPLEX 10.1 Reference Manual, 2006.
24. P. Kilby, J. K. Slaney, S. Thibaux, and T. Walsh. Backbones and backdoors in satisfiability. In *Proceedings of AAAI-05: 20th National Conference on Artificial Intelligence*, pages 1368–1373, 2005.
25. C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of IJCAI-97: 15th International Joint Conference on Artificial Intelligence*, pages 366–371, Nagoya, Japan, Aug. 1997.
26. I. Lynce and J. P. Marques-Silva. Hidden structure in unsatisfiable random 3-SAT: An empirical study. In *ICTAI-06: 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 246–251, Boca Raton, FL, Nov. 2004.
27. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of DAC-01: 38th Design Automation Conference*, pages 530–535, Las Vegas, NV, June 2001.
28. N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proceedings of SAT-04: 7th International Conference on Theory and Applications of Satisfiability Testing*, Vancouver, BC, Canada, May 2004.
29. N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. In *Proceedings of SAT-06: 9th International Conference on Theory and Applications of Satisfiability Testing*, pages 396–409, 2006.
30. L. Paris, R. Ostrowski, P. Siegel, and L. Sais. Computing Horn strong backdoor sets thanks to local search. *ICTAI-06: 17th IEEE International Conference on Tools with Artificial Intelligence*, pages 139–143, Nov. 2006.
31. M. Samer and S. Szeider. Constraint satisfaction with bounded treewidth revisited. In *CP-06: 12th International Conference on Principles and Practice of Constraint Programming*, pages 499–513, Nantes, France, 2006.
32. C. Sinz, A. Kaiser, and W. Küchlin. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(1): 75–97, Jan. 2003. Special issue on configuration.
33. S. Szeider. Backdoor sets for DLL subsolvers. *Journal of Automated Reasoning*, 35(1-3):73–88, 2005.
34. P. van Beek and R. Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, 42(3):543–561, 1995.
35. R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI-03: 18th International Joint Conference on Artificial Intelligence*, pages 1173–1178, Acapulco, Mexico, Aug. 2003.

36. R. Williams, C. Gomes, and B. Selman. On the connections between heavy-tails, backdoors, and restarts in combinatorial search. In *Proceedings of SAT-03: 6th International Conference on Theory and Applications of Satisfiability Testing*, pages 222–230, Santa Margherita Ligure, Italy, May 2003.

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 $\mathcal{G}$ in which the actions of a player and his payoff depend on the actions of a subset of the other players. $\mathcal{G}$ 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 $\mathcal{G}$ 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 $\mathcal{G}$ has a PNE as a CNF formula F that is satisfiable if and only if $\mathcal{G}$ has a PNE.

For every player p in $\mathcal{G}$ 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.[3] Further, let the neighbors of p be $(q_1, q_2, \ldots, q_k)$, in some arbitrary but fixed order. As discussed above, the payoffs in $\mathcal{G}$ determine a set of best-response actions $BR_p(a_1, a_2, \ldots, a_k)$ for p when the action of player q_i is a_i, $1 \leq i \leq k$. Therefore, F must encode the fact that when

[3] 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.

literals $x_{q_i}^{a_i}$ are True for all $1 \leq i \leq k$, then literals x_p^b are False for all non-best-response actions of p, i.e., for all $b \in \text{actions}(p) \setminus BR_p(a_1, a_2, \ldots, a_k)$.

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

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

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

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

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

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

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