Tradeoffs in Backdoors: Inconsistency Detection, Dynamic Simplification, and Preprocessing *

Bistra Dilkina and **Carla P. Gomes** and **Ashish Sabharwal**
Department of Computer Science, Cornell University, Ithaca, NY 14853, U.S.A.
{bistra,gomes,sabhar}@cs.cornell.edu

Abstract

There has been considerable interest in the identification of structural properties of combinatorial problems that lead to efficient algorithms for solving them. The notion of backdoors captures hidden structure exploited by state-of-the-art constraint solvers. We prove that strong backdoor identification with respect to the tractable classes Horn and 2CNF 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 such as car configuration and logistics that have thousands of variables often have backdoors of only a few variables. We evaluate the effect of different preprocessors on the structure of the instances and hence on the backdoor size. Finally, strong backdoors have been mostly studied for unsatisfiable instances. In this work, we also look into strong backdoors of satisfiable instances and their relationship to solution counting.

1 Introduction

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, e.g. 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 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 the topology of the underlying constraint graph.

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 (Williams, Gomes, & Selman, 2003a,b; Gomes *et al.*, 2000; Chen, Gomes, & Selman, 2001; Kilby *et al.*, 2005; Szeider, 2005). Note that the notion of tractability in the definition of backdoor sets is not necessarily syntactically defined: it may be defined 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 SAT solvers 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.

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 w.r.t. a particular tractable class if backdoor detection w.r.t. that class is shown to be fixed parameter tractable. Indeed, Nishimura, Ragde, & Szeider (2004) showed that detecting strong backdoors (cf. Section 2 for a formal definition) w.r.t. the classes 2CNF and Horn is NP-complete but also fixed-parameter tractable. Note, however, that this result is only w.r.t. the tractable classes of *pure* 2CNF/Horn. In particular, certain kinds of obvi-

*Some of the material in this paper was presented at the CP-07 conference (Dilkina, Gomes, & Sabharwal, 2007).

ous inconsistencies are not detected in these classes, such as having an empty clause in an *arbitrary* formula — clearly, any basic solver detects such inconsistencies. 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 the seemingly small feature of 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, 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, we also consider backdoors w.r.t. RHorn (renamable Horn), UP (unit propagation), PL (pure literal rule), UP+PL, 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.

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. 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* (Li & Anbulagan, 1997; Gomes, Selman, & Kautz, 1998) is remarkably good at finding small strong backdoors on a range of experimental domains. For example, for the car configuration problem, strong "SATZ-backdoor" sets involve 0-0.7% 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 instances.

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 Kilby *et al.* (2005); Dilkina, Gomes, & Sabharwal (2007). In this work, we look at strong backdoor sizes in some satisfiable instances, 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.

2 Preliminaries and Related Work

A *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.

The concept of backdoors and their theoretical foundations were introduced by Williams, Gomes, and Selman (Williams, Gomes, & Selman, 2003a,b). 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. 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. 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 $\tau : B \to \{0,1\}$, 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 $\tau : B \to \{0,1\}$, S correctly determines $F[\tau/B]$?" We are interested in the complexity of this problem for specific sub-solvers.

Although we focus on strong backdoors, a related notion applicable to satisfiable formulas only is that of "weak" backdoors. The notion of weak backdoors with respect to satisfiable formulas captures, in a sense, a "witness" to the satisfiability of the instance. It requires that the sub-solver is able to find a solution for the formula, given *some* assignment to the weak backdoor variables (as opposed to all assignments).

Definition 3. A set B of variables is a *weak backdoor set* for a formula F w.r.t. a sub-solver S if $B \subseteq var(F)$ and for some truth assignment $\tau : B \rightarrow \{0,1\}$, S returns a satisfying assignment for $F[\tau]$.

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 & Marques-Silva (2004) show that the search effort required by the SAT solver *zChaff* (Moskewicz *et al.*, 2001) 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. A *unit clause* is a clause that contains only one literal. 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. 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. 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.

Szeider (2005) studied the complexity of finding strong backdoors w.r.t. the above 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 C-backdoor of size k is complete for the parameterized complexity class W[P]. Interestingly, the naïve brute-force procedure for this problem w.r.t. any sub-solver S 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, 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 (Li & Anbulagan, 1997) 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.* (2005) 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.

3 Two Key Properties of Backdoors

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 each 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 to the efficiency of SAT solvers. In this section, we show that while relaxing each of these properties can make identifying backdoors computationally easier, it also can lead to exponentially larger backdoor sizes and hence weakens the usefulness of the notion of backdoors. We review the theoretical results of (Dilkina, Gomes, & Sabharwal, 2007), omitting complete proof details.

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 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. 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. We will refer to backdoors w.r.t. to these tractable classes 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 trivially 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 4. $C_{\{\}}$ 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 show that strong backdoors w.r.t. 2CNF$^{\{\}}$ and 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. In particular, the two problems of deciding whether a formula has

a strong backdoor w.r.t. 2CNF$^{\{\}}$ and 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 (Nishimura, Ragde, & Szeider, 2004). Recall that adding $C_{\{\}}$ to 2CNF and Horn corresponds to adding empty clause detection to the two classes.

Let $C \in \{\text{Horn, 2CNF}\}$. Given a formula F and $k \geq 0$, we show that the problem of deciding whether F has a strong $C^{\{\}}$-backdoor of size k is NP-hard by extending the argument originally used by Nishimura, Ragde, & Szeider (2004) for (pure) 2CNF/Horn using a reduction from the NP-complete Vertex Cover problem. UNSAT is the coNP-complete problem of deciding whether a given CNF formula is unsatisfiable. We prove coNP-hardness of backdoor detection w.r.t. Horn$^{\{\}}$ and 2CNF$^{\{\}}$ by reducing UNSAT to strong $C^{\{\}}$-backdoor detection exploiting the fact that Horn and 2CNF are closed under clause deletion. Combining the two results together gives us our main theorem:

Theorem 1. *Let $C \in \{\text{Horn, 2CNF}\}$. 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.*

Although the seemingly small feature of empty clause detection results in an increase in formal complexity of backdoor detection, it also has a clear positive aspect in that adding empty clause detection to Horn and 2CNF can result in arbitrarily smaller minimum backdoor size. Let us consider the following formula $F = (x_0) \wedge (\neg x_0) \wedge (x_0 \vee x_1 \ldots \vee x_n)$. The variable x_0 clearly corresponds to a a strong $C^{\{\}}$-backdoor of size 1, while the smallest Horn backdoor has size n.

A different notion of backdoors, motivated by the work of Nishimura, Ragde, & Szeider (2004), 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). The *deletion* of a variable x from a formula F corresponds to syntactically removing the literals of x from F: $F - x = \{c \setminus \{x^0, x^1\} \mid c \in F\}$. For $X \subseteq var(F)$, $F - X$ is defined similarly.

Definition 5 (deletion C-backdoor). 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 (Nishimura, Ragde, & Szeider, 2004), Horn (Nishimura, Ragde, & Szeider, 2004), or RHorn (Chandru & Hooker, 1992).

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 $C \in \{\text{2CNF, Horn}\}$, deletion and strong backdoors are equivalent, a key fact underlying the fixed parameter algorithm of Nishimura, Ragde, & Szeider (2004). We will show that this equivalence between deletion backdoors and strong backdoors does not hold for RHorn.

We prove that strong RHorn-backdoors can be exponentially smaller than "static" deletion RHorn-backdoors, and are therefore more likely to succinctly capture structural properties of interest in formulas. The main idea 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 for the variables in $F[\tau/B]$ such that $F[\tau/B]$ under the renaming r_τ yields a Horn formula. If F is carefully constructed 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.

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

4 Strong Backdoor Sizes 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 Kilby *et al.* (2005); Lynce & Marques-Silva (2004). In this section, we demonstrate that, although one cannot efficiently identify minimum backdoor sets, in practice many real-world combinatorial problems have surprisingly small backdoors.

We analyze the size of strong and deletion backdoors w.r.t. several classes and sub-solvers in four problem domains.

We consider backdoors w.r.t. the tractable classes Horn and RHorn. The problems of finding a smallest deletion Horn-backdoor (equivalent to strong Horn-backdoor) and of finding a smallest deletion RHorn-backdoor can be formulated as 0-1 integer programs (omitted here). Using such encodings and the ILOG CPLEX libraries (ILOG, SA, 2006) we compute optimal (smallest) Horn- and RHorn-backdoors in our experimental evaluation of backdoor size.

Following previous work (Williams, Gomes, & Selman, 2003a; Kilby *et al.*, 2005), we also consider strong SATZ-backdoors. We obtain an upper bound on the size of the smallest strong SATZ-backdoor by running *SatzRand* (Gomes, Selman, & Kautz, 1998) (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 which includes UP, PL and probing. Hence, this set of variables is a backdoor w.r.t. to the propa-

instance set	num vars	num clauses	Horn %	RHorn (del) %	SATZ %	UP+PL %	UP %
gcp_300	900	67724.4	66.67	16.78	0.11	0.51	0.60
gcp_500	1500	187556.0	66.67	16.73	0.07	0.28	0.80
map_30_57	13424	103120	48.21	48.19	0	0.41	3.23
map_50_97	38364	438840	48.93	48.92	0	0.25	3.19
pne	2000	40958.9	67.88	66.86	0.05	0.38	0.42
pne	5000	98930.8	67.80	66.80	0.00	0.13	0.15
C168_FW_SZ	1698	5646.8	14.32	2.83	0.16	0.77	5.70
C210_FW_RZ	1789	7408.3	12.54	4.81	0.65	1.42	12.97
C210_FW_SZ	1789	7511.8	13.74	5.37	0.23	0.78	11.15
C220_FV_SZ	1728	4758.2	9.14	2.92	0.19	0.46	8.88

Table 1: Strong backdoor sizes for Graph Coloring (gcp), MAP planning (map), Pure Nash Equilibrium (pne), and Car Configuration (Cxxx). Each row reports the average over several instances. Backdoor sizes are shown as the % of the number of problem variables. The RHorn numbers are for deletion backdoors. Horn- and RHorn-backdoor sizes are the smallest sizes, while the rest are upper bounds.

gation mechanism of *Satz*(i.e. a SATZ-backdoor) and its size is an upper bound on the minimum SATZ-backdoor size.

We also record the set of variables not set by UP and PL while searching with *Satz*. By a similar reasoning as above, the size of this set gives us an upper bound on the smallest strong (UP+PL)-backdoor size. Similarly, we record all variables set in *Satz-Rand* by anything but the UP(or PL) procedure to obtain an upper bound on the smallest strong UP(or PL)-backdoor size. For satisfiable instances, we obtain upper bounds on the smallest backdoors by forcing *Satz* 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 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 backdoor size for this instance.

For our experimental evaluation, we considered four problem domains: graph coloring, logistics planning, equilibrium problems from game theory, and car configuration. The results are shown in Table 1 (more results are available in (Dilkina, Gomes, & Sabharwal, 2007)).

We generated graph coloring instances using the clique hiding graph generator of Brockington & Culberson (1996) 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. For problems with 500 graph nodes, the $C_{\{\}}$-backdoor comprises of only 0.8% of the variables. This domain illustrates how strong Horn-backdoors and deletion RHorn-backdoors can be significantly larger than backdoors w.r.t. empty clause detection; it also shows that deletion RHorn-backdoors (involving $\approx$ 17% of the variables) are considerably smaller than strong Horn-backdoors ($\approx$ 67%). We note that *Satz* finds backdoors even smaller than the $C_{\{\}}$-backdoor.

The MAP problem domain is a synthetic logistics planning domain for which the size of the strong UP-backdoors is well understood (Hoffmann, Gomes, & Selman, 2007). All MAP instances considered are unsatisfiable, encoding one planning step less than the length of the optimal plan. Hoffmann, Gomes, & Selman (2007) identify that certain MAP instances (called asymmetric) have logarithmic size DPLL refutations (and backdoors) which we consider here. In this 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, implying that the smallest strong SATZ-backdoor is of size 0.

The game theory instances encode the problem of deciding the existence of an equilibrium strategy in a graphical game. Here we consider binary games, where each player has exactly two action choices, with payoff values generated independently u.a.r. and interaction graphs that are drawn from the Erdös-Rényi random graph model $\mathbb{G}(n,p)$. For this domain, while strong Horn-backdoor sets and deletion RHorn-backdoor involve $\approx$ 68% and $\approx$ 67% of the variables, respectively, strong SATZ-backdoors are surprisingly small, close to 0% of the variables.

Finally, we also consider a real-world SAT benchmark from product configuration. The instances encode problems from the validation and verification of automotive product configuration data for the Daimler Chrysler's Mercedes car lines (Sinz, Kaiser, & Küchlin, 2003). We consider a set of unsatisfiable instances available at http://www-sr.informatik.uni-tuebingen.de/∼sinz/DC/. Here, while strong Horn-backdoors vary between 10-25% of the variables, RHorn-backdoor sets are considerably smaller at 3-8%. Strong SATZ-backdoors involve only 0-0.7% of the variables.

5 Effect of Preprocessors

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* (Li & Anbulagan, 1997). It resolves clauses of length at most 3 until saturation. 2-SIMPLIFY (Brafman, 2001) 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 (Bacchus & Winter, 2003) reasons on binary clauses similarly to 2-SIMPLIFY, but also incorporates full resolution. Also, unit propagation and equality reduction are applied until saturation. SatELite (Eén & Biere, 2005) 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 (Hoos & Stützle, 2000), 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.

File	vars	clauses	Satz	UP+PL
bf0432-007.cnf	1040	3122	80	217
2-SIMPLIFY	1205	2501	67	168
HyPre	325	1383	65	156
SatELite	556	2216	132	219
3-Resolution	795	3499	30	191
bf1355-075.cnf	2180	5146	44	117
2-SIMPLIFY	2583	4134	179	221
HyPre	704	3124	82	103
SatELite	0	0	0	0
3-Resolution	1682	5557	72	158
bf1355-638.cnf	2177	5385	34	142
2-SIMPLIFY	2589	4861	104	152
HyPre	486	2010	37	50
SatELite	814	3134	0	5
3-Resolution	1461	4397	46	81
bf2670-001.cnf	1393	2926	16	32
2-SIMPLIFY	1592	1629	8	29
HyPre	0	0	0	0
SatELite	80	294	0	10
3-Resolution	1202	2970	7	19

Table 2: 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.

6 Backdoors for Satisfiable Instances

Most of the previous work on strong backdoors has analyzed unsatisfiable instances (e.g. Lynce & Marques-Silva, 2004; Dilkina, Gomes, & Sabharwal, 2007). 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 (Sinz, Kaiser, & Küchlin, 2003). 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" (Gomes & Sellmann, 2004; Gomes, Sabharwal, & Selman, 2006): as PL assigns variables that appear as pure literals, it guarantees that the satisfiability of the formula remains unchanged, while at the same time potentially pruning several solutions.

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

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.

7 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,[1] 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 $\#F$.

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

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 propagation subsolver. 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 Nishimura, Ragde, & Szeider (2006). 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.

8 Conclusions

The complexity of finding backdoors is influenced significantly by the features of the underlying sub-solver or tractable problem class. In particular, strong backdoor identification w.r.t. to Horn and 2CNF becomes harder than NP (unless NP=coNP) as soon as the seemingly small feature of empty clause detection is incorporated, but in practice it reduces the size of the backdoors dramatically. We show that in addition to inconsistency detection, also the dynamic constraint propagation included in the definition of strong backdoor is key to capturing small backdoor sets. For the class RHorn, we prove that "static" deletion backdoors can be exponentially larger than strong backdoors, in contrast with the known results for 2CNF- and 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, 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.

Acknowledgments

This research was supported by IISI, Cornell University, AFOSR Grant FA9550-04-1-0151.

References

Bacchus, F., and Winter, J. 2003. Effective preprocessing with hyper-resolution and equality reduction. In *SAT*, 341–355.

Brafman, R. I. 2001. A simplifier for propositional formulas with many binary clauses. In *IJCAI*, 515–522.

Brockington, M., and Culberson, J. C. 1996. Camouflaging independent sets in quasi-random graphs. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26, 75–88. American Mathematical Society.

Chandru, V., and Hooker, J. N. 1992. Detecting embedded Horn structure in propositional logic. *Information Processing Letters* 42(2):109–111.

Chen, H.; Gomes, C.; and Selman, B. 2001. Formal models of heavy-tailed behavior in combinatorial search. In *CP'01*.

Dilkina, B.; Gomes, C. P.; and Sabharwal, A. 2007. Tradeoffs in the complexity of backdoor detection. In *Principles and Practice of Constraint Programming - CP 2007*, 256–270.

Eén, N., and Biere, A. 2005. Effective preprocessing in sat through variable and clause elimination. In *SAT*, 61–75.

Gomes, C. P., and Sellmann, M. 2004. Streamlined constraint reasoning. In *CP*, 274–289.

Gomes, C. P.; Selman, B.; Crato, N.; and Kautz, H. 2000. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.* 24(1-2):67–100.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model counting: A new strategy for obtaining good bounds. In *AAAI*.

Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting Combinatorial Search Through Randomization. In *AAAI'98*, 431–438.

Hoffmann, J.; Gomes, C.; and Selman, B. 2007. Structure and problem hardness: Goal asymmetry and DPLL proofs in SAT-based planning. *Logical Methods in Computer Science* 3(1:6).

Hoos, H. H., and Stützle, T. 2000. SATLIB: An Online Resource for Research on SAT. In *SAT'00*. 283–292.

ILOG, SA. 2006. CPLEX 10.1 Reference Manual.

Kilby, P.; Slaney, J. K.; Thibaux, S.; and Walsh, T. 2005. Backbones and backdoors in satisfiability. In *AAAI'05*, 1368–1373.

Li, C. M., and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability problems. In *IJCAI'97*, 366–371.

Lynce, I., and Marques-Silva, J. 2004. Hidden structure in unsatisfiable random 3-SAT: An empirical study. In *ICTAI'04*.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: engineering an efficient SAT solver. In *DAC'01*, 530–535.

Nishimura, N.; Ragde, P.; and Szeider, S. 2004. Detecting backdoor sets with respect to Horn and binary clauses. In *SAT'04*.

Nishimura, N.; Ragde, P.; and Szeider, S. 2006. Solving #SAT using vertex covers. In *SAT'06*, 396–409.

Sinz, C.; Kaiser, A.; and Küchlin, W. 2003. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engr. Design, Analysis and Manufacturing* 17(1):75–97. Special issue on configuration.

Szeider, S. 2005. Backdoor sets for dll subsolvers. *J. Autom. Reason.* 35(1-3):73–88.

Williams, R.; Gomes, C.; and Selman, B. 2003a. Backdoors to typical case complexity. In *IJCAI'03*, 1173–1178.

Williams, R.; Gomes, C.; and Selman, B. 2003b. On the connections between heavy-tails, backdoors, and restarts in combinatorial search. In *SAT'03*, 222–230.