Backdoors for SAT

Marco Gario
Supervisor: Steffen Hölldobler
Advisor: Norbert Manthey

EMCL / TUD

February 21, 2012
Content

1 Introduction
2 Backdoors
3 Experimental Results
1 Introduction

2 Backdoors

3 Experimental Results
A few auxiliary definitions:

- Class $C$: a set of formulas sharing some property.
- A CNF formula $F$ is in *Horn* iff each of its clauses has *at most* one positive literal.
- $F$ is in Horn:

$$F = (x \lor \neg y) \land (y \lor \neg z \lor \neg w) \land (\neg w \lor \neg z)$$

- Horn can be solved in polynomial time (e.g., by unit propagation)
Assume I give you a CNF like this (1000 clauses, >100 variables, 3-SAT):

\[(x_1 \lor \neg x_2 \lor \neg x_3) \lor \ldots \lor (x_{n_1} \lor \neg x_{n_2} \lor \neg x_{n_3}) \lor (x_{m_1} \lor x_{m_2} \lor x_{m_3})\]

- How difficult is this instance? (E.g., could you solve it by hand?)
- Solving instances of this kind gives me an NP-hard problem?
- This is FPT!
Assume I give you a CNF like this (1000 clauses, >100 variables, 3-SAT):

\[(x_1 \lor \neg x_2 \lor \neg x_3) \lor\]
\[\vdots \text{997 Horn clauses}\ldots \lor\]
\[(x_{n_1} \lor \neg x_{n_2} \lor \neg x_{n_3}) \lor\]
\[(x_{m_1} \lor x_{m_2} \lor x_{m_3})\]

- How difficult is this instance? (E.g., could you solve it by hand?)
- Solving instances of this kind gives me an NP-hard problem?
- This is FPT!
A backdoor is a set of variables. Once we assign a value to the backdoor variables, the problem becomes tractable (in P). Introduced by Williams et al. ([8]) to try to explain the good performances of modern SAT solvers. Their claim: modern SAT solver can find backdoors easily. But solvers are NOT designed for this! Why don’t we try to pro-actively find these backdoors? Finding backdoors is an NP-Hard problem!
Previous work

Theoretical work:
- Defining several types of backdoors,
- Complexity of finding them (especially parameterized complexity)

Empirical work:
- Showing that backdoor sets are “small”, for different types of backdoors
  - Results are mostly based on local search algorithms! (Incompleteness)
- Little information on runtime required to find them
- One work ([5]) shows that using Horn-backdoors improves SAT solving speed. However, no information on how long it takes to find them!

Questions:
→ How can we find backdoors efficiently?
→ Can we “predict” backdoors by using additional domain knowledge?
Outline

1. Introduction
2. Backdoors
3. Experimental Results
$F - B$: Replacing in each clause of $F$ the occurrences of $x$ and $\neg x$ with $\bot$ for each variable $x \in B$ (with $B \subseteq \text{var}(F)$) and simplifies the clause. Basically, remove the occurrences (positive and negative) of the variables in $B$ from $F$.

**Definition: Deletion $C$-backdoor ([4])**

A non-empty subset $B$ of the variables of the formula $F$ ($B \subseteq \text{var}(F)$) is a *deletion backdoor* w.r.t. a class $C$ for $F$ iff $F - B \in C$.

**Example**

\[
F = (x \lor y \lor z) \land (\neg x \lor z \lor w) \land (y \lor w) \quad B = \{z, y\}
\]

\[
F - B \equiv (x \lor y \lor \neg z) \land (\neg x \lor \neg z \lor w) \land (y \lor w)
\]
Deletion Backdoors (1/2)

$F - B$: Replacing in each clause of $F$ the occurrences of $x$ and $\neg x$ with $\bot$ for each variable $x \in B$ (with $B \subseteq \text{var}(F)$) and simplifies the clause. Basically, remove the occurrences (positive and negative) of the variables in $B$ from $F$.

**Definition: Deletion C-backdoor ([4])**

A non-empty subset $B$ of the variables of the formula $F$ ($B \subseteq \text{var}(F)$) is a deletion backdoor w.r.t. a class $C$ for $F$ iff $F - B \in C$.

**Example**

$$F = (x \lor y \lor z) \land (\neg x \lor z \lor w) \land (y \lor w) \quad B = \{z, y\}$$

$$F - B \equiv x \land (\neg x \lor w) \land w$$
From a deletion $C$-backdoor we can generate $2^{|B|}$ formulas, each with a different assignment to the backdoor variables. Once the backdoor is decided, we can solve these instances in polynomial time!
Definition: Vertex Cover

Given a graph $G = (V, E)$, we call $R = \{v_1, \ldots, v_n\} \subseteq V$ a vertex cover of $G$ iff for all $e \in E$ there exists a $v_i \in R$ s.t. $v_i \in e$.

In other words, we have a vertex $v_i \in R$ as representative of each edge in $E$. We call $|R|$ the size of the vertex cover.

Definition: Vertex Cover Problem

Given a graph $G = (V, E)$ and an integer $k > 0$, is there a vertex cover $R$ for $G$ s.t. $|R| \leq k$ ?
Example

Consider $G = (V, E)$:

Then $C_1 = \{v_1, v_4\}$ and $C_2 = \{v_2, v_3, v_4\}$ are vertex covers for $G$ but only $C_1$ is a solution for the Vertex cover instance $(G, 2)$. 
Samer and Szeider ([7]) propose a reduction from deletion Horn-backdoor detection to the vertex cover problem.

**Definition**

$G_F$ is the graph composed by the variables of the CNF formula $F$ in which two variables $v, u$ are adjacent iff $v$ and $u$ appear positively in a clause from $F$.

**Lemma**

A set $B \subseteq \text{var}(F)$ is a deletion Horn-backdoor for $F$ iff $B$ is a vertex cover of $G_F$.

This relation extends also to Minimum Vertex Cover, in which we are interested in the vertex cover/backdoor of minimal size (smallest backdoor).
Example

\[ F = (x \lor y \lor z) \land (\neg x \lor z \lor w) \land (y \lor w) \]

\[ F - C \equiv x \land (\neg x \lor w) \land w \]

We can use existing results from Vertex Cover (including FPT results) to solve deletion Horn-backdoor detection!
Goals & Challenges

- Study deletion Horn-backdoors in SAT instances
  - Build a dataset *efficiently*, e.g. FPT
    - No available implementation of parameterized Vertex Cover
  - Implement algorithms that are a good trade-off between performance and implementation complexity
- Test whether we can use local search to efficiently find smallest deletion Horn-backdoors
- Study the relation of the backdoor size with features of the instances
Goals & Challenges

- Study deletion Horn-backdoors in SAT instances
  - Build a dataset \textit{efficiently}, e.g. FPT
  - No available implementation of parameterized Vertex Cover
  - Implement algorithms that are a good trade-off between performance and implementation complexity
- Test whether we can use local search to efficiently find smallest deletion Horn-backdoors
- Study the relation of the backdoor size with features of the instances
Goals & Challenges

- Study deletion Horn-backdoors in SAT instances
  - Build a dataset *efficiently*, e.g. FPT
  - No available implementation of parameterized Vertex Cover
  - Implement algorithms that are a good trade-off between performance and implementation complexity
- Test whether we can use local search to efficiently find smallest deletion Horn-backdoors
- Study the relation of the backdoor size with features of the instances
Solving Vertex Cover

- Local search algorithm: COVER ([6])
- FPT algorithms: Kernelization and Bounded search
- Reduction to SAT
Outline

1. Introduction
2. Backdoors
3. Experimental Results
Methodology

1. Benchmark with 3239 instances from various sources;
2. Generate the associated vertex cover instances;
3. Run a modified version of COVER [6] to obtain an upper-bound on the size of the smallest deletion Horn-backdoor;
4. For instances with small backdoors ($k \leq 150$) verify the minimality of the backdoor;
5. For instances with bigger backdoors confirm that the lower-bound is bigger than 150;
6. Considering only the instances for which we have the exact value of the smallest backdoor, compute the quality of the solution provided by a fast version of COVER.
Results

2418 (74%) instances have upper-bound on the size of the deletion Horn-backdoor up to 150:

- 2357 (97.5%) verified by the FPT algorithms,
- 8 more with CryptoMinisat,
- 53 remain unverified

Runtime\(^1\)

- Time-out: 90 minutes
- Average: 25 second; 87% in less than 5 seconds, 93% in under a minute (thanks to kernelization!)

The generated vertex cover instances were in most of the cases easy, but a few were really hard (2%).

\(^1\)Timings based on Intel Centrino 1.7Ghz, 1GB RAM
We define two configurations for COVER.

Full computation:
- Runtime: 30-90 minutes
- Solution quality: Always finds the optimum

Fast computation:
- Runtime: 115 ms (avg), 97 ms (avg) for $k \leq 150$
- Solution quality: 98% of the times optimum

Average error among all the instances is 0.11%.

COVER is a good method to compute smallest deletion Horn-backdoors.
We are interested in the relation between smallest deletion Horn-backdoor (sdH-bd) size and other properties of the instances:

**Flat**  Colouring on flat graphs; sdH-bd size is exactly two times the number of vertices in the graph.

**Random**  Uniform random formulas (uf/uuf): correlation between number of variables and sdH-bd size is 0.99.

**CarConf**  Verify some consistency properties of requested car configuration. Correlation between number of variables and upper-bound of the sdH-bd size for the same configuration is high:

<table>
<thead>
<tr>
<th>Base configuration</th>
<th>Correlation (r)</th>
<th># Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>C168</td>
<td>0.99</td>
<td>58</td>
</tr>
<tr>
<td>C170</td>
<td>0.99</td>
<td>6</td>
</tr>
<tr>
<td>C202</td>
<td>0.83</td>
<td>23</td>
</tr>
<tr>
<td>C208</td>
<td>0.99</td>
<td>16</td>
</tr>
<tr>
<td>C210</td>
<td>0.89</td>
<td>32</td>
</tr>
<tr>
<td>C220</td>
<td>0.95</td>
<td>348</td>
</tr>
<tr>
<td>C638</td>
<td>0.73</td>
<td>84</td>
</tr>
</tbody>
</table>
Conclusions

We studied the relation between deletion Horn-backdoor detection and the vertex cover problem.

- COVER is a good way of computing quickly (and with an excellent quality) smallest deletion Horn-backdoors.
- Kernelization can play a key role. Even with a simple kernelization, we solved many instances without search.
- In some cases, features of an instance can be related with the size of its smallest deletion Horn backdoor.
Further work

- Implement COVER/deletion Horn-backdoor detection in a solver and allow branching only on backdoors variables;
- Use backdoors to explore different solver architectures, and not only DPLL;
- Influence of preprocessing on different classes of backdoors;
- Build predictive models that can find backdoors!
Questions?

Datasets, tools and slides are available online: http://marco.gario.org/work/
Kernelization algorithms for the vertex cover problem: Theory and experiments.

Y. Crama, O. Ekin, and P.L. Hammer.
Variable and term removal from Boolean formulae.

F. Hüffner, R. Niedermeier, and S. Wernicke.
Techniques for Practical Fixed-Parameter Algorithms.


Ryan Williams, C.P. Gomes, and Bart Selman.
Backdoors to typical case complexity.
For an NP-Hard problem we have an exponential worst-case runtime in the size of the problem: e.g. SAT $O(2^n)$

We can do better by defining a parameter $k$ on which to confine the exponential explosion: e.g. p-SAT $O(2^k \times n^c)$, with $k$ number of variables and some constant $c$

Problems for which we can identify such parameters are called *Fixed-parameter tractable* (FPT)

Note that SAT parameterized by the size of a backdoor is FPT ($O(2^{|B|} \times n^c)$)
We implement a simple bounded search by Hüffner ([3]),

The *trivial* solution for vertex cover has complexity $O(2^k)$

The *best* $O(1.2738^k)$

This one $O(1.47^k)$
Kernelization is important:

- 42% of all instances were solved by kernelization (51.5% in the group with $k \leq 150$);
- Otherwise, parameter reduction of 17.8% (avg)
- Parameter has exponential influence on runtime
- Extreme case from $k = 109$ to $k' = 6$

219 instances might have a solution of size $\leq 150$, but remain unverified.
$F$: a propositional formula in CNF

$\text{var}(F)$: the set of variables occurring in $F$.

$l, \overline{l}$: a positive variable or its negation

$J$: (partial) interpretation. Partial mapping from $\text{var}(F)$ to the boolean values $\{\top, \bot\}$.

Class $C$: a set of formulas sharing some property.

Horn class: $F$ is in Horn iff each of its clauses has at most one positive literal.
Example

- $F = a \land \neg b \land (\neg d \lor c) \land (\neg c \lor d)$
- $\text{var}(F) = \{a, b, c, d\}$
- $J = \{\neg b, c\}$
- $F|_J \equiv a \land \neg b \land (\neg d \lor \neg c) \land (\neg c \lor d) \equiv a \land d$
- For $V = \{c\}$,
  - $F - V \equiv a \land \neg b \land (\neg d \lor \neg c) \land (\neg c \lor d) \equiv a \land \neg b \land \neg d \land d$
- $F \in \text{Horn}$
Subsolvers

Definition: Subsolver [8]

We call an algorithm $C$ a subsolver if, given an input formula $F$:

**Tricotomy:** $C$ either rejects the input $F$, or “determines” $F$ correctly (as unsatisfiable or satisfiable, returning a solution if satisfiable),

**Efficiency:** $C$ runs in polynomial time,

**Trivial solvability:** $C$ can determine if $F$ is trivially true (has no constraints) or trivially false (has contradictory constraint),

**Self-reducibility:** if $C$ determines $F$, then for any assignment $J$ of the variable $x$, $C$ determines $F|_J$

There exists a subsolver for Horn! And also for other classes: e.g. RHorn, 2SAT, UP+PL.
Definition: Subsolver [8]
We call an algorithm $C$ a subsolver if, given an input formula $F$:

- **Tricotomy**: $C$ either rejects the input $F$, or “determines” $F$ correctly (as unsatisfiable or satisfiable, returning a solution if satisfiable),
- **Efficiency**: $C$ runs in polynomial time,
- **Trivial solvability**: $C$ can determine if $F$ is trivially true (has no constraints) or trivially false (has contradictory constraint),
- **Self-reducibility**: if $C$ determines $F$, then for any assignment $J$ of the variable $x$, $C$ determines $F|_J$

There exists a subsolver for Horn! And also for other classes: e.g. RHorn, 2SAT, UP+PL.
For preprocessing we use the following four rules:

**P1** A vertex of degree 0 cannot be part of any cover, therefore we obtain $G'$ by removing it from $G$.

**P2** If there is a vertex $x$ of degree 1, then there is an optimal vertex cover in which its neighbour $y$ is in the cover.
P3 If there is a vertex of degree 2 with two adjacent neighbours, then there is an optimal vertex cover containing both these neighbours.

P4 If there is a vertex $x$ of degree 2 with two non-adjacent neighbours $y$ and $w$, then $x$ can be removed by contracting the edges $(w, x)$ and $(x, y)$. 
We use this simple bounded search by Hüffner ([3])

**S1** If there is a vertex of degree one, put its neighbour into the cover.

**S2** If there is a vertex $x$ of degree two, then either i) both neighbours of $x$ are in an optimal vertex cover, or ii) $x$ is in an optimal cover together with all neighbours of its neighbours.
Bounded search II

First branch

Second branch
S3 If there is a vertex $x$ of degree at least three, then either $x$ or all its neighbours are in the cover.
**Strong/Deletion Backdoors (1/2)**

\( F_J \): reduct of \( F \) w.r.t. the (partial) interpretation \( J \); it is obtained by replacing each variable \( v \) in \( F \) with \( J(v) \) and simplifying.

**Definition: Strong C-Backdoor ([8])**

A non-empty subset \( B \) of the variables of the formula \( F \) (\( B \subseteq \text{var}(F) \)) is a **strong backdoor** w.r.t. the subsolver \( C \) for \( F \) iff for all interpretations \( J : B \rightarrow \{\top, \bot\} \), \( C \) returns a satisfying assignment or concludes unsatisfiability of \( F_J \).

\( F - V \): replaces in each clause of \( F \) the occurrences of \( x \) and \( \neg x \) with \( \bot \) for each variable \( x \in V \) (with \( V \subseteq \text{var}(F) \)) and simplifies the clause.

**Definition: Deletion C-backdoor ([4])**

A non-empty subset \( B \) of the variables of the formula \( F \) (\( B \subseteq \text{var}(F) \)) is a **deletion backdoor** w.r.t. a class \( C \) for \( F \) iff \( F - B \in C \).
**Strong/Deletion Backdoors (1/2)**

\(F|_J:\) reduct of \(F\) w.r.t. the (partial) interpretation \(J\); it is obtained by replacing each variable \(v\) in \(F\) with \(J(v)\) and simplifying.

**Definition: Strong C-Backdoor ([8])**

A non-empty subset \(B\) of the variables of the formula \(F\) (\(B \subseteq \text{var}(F)\)) is a strong backdoor w.r.t. the subsolver \(C\) for \(F\) iff for all interpretations \(J : B \rightarrow \{\top, \bot\}\), \(C\) returns a satisfying assignment or concludes unsatisfiability of \(F|_J\).

\(F - V:\) replaces in each clause of \(F\) the occurrences of \(x\) and \(\neg x\) with \(\bot\) for each variable \(x \in V\) (with \(V \subseteq \text{var}(F)\)) and simplifies the clause.

**Definition: Deletion C-backdoor ([4])**

A non-empty subset \(B\) of the variables of the formula \(F\) (\(B \subseteq \text{var}(F)\)) is a deletion backdoor w.r.t. a class \(C\) for \(F\) iff \(F - B \in C\).
$F|_J$: reduct of $F$ w.r.t. the (partial) interpretation $J$; it is obtained by replacing each variable $v$ in $F$ with $J(v)$ and simplifying.

**Definition: Strong $C$-Backdoor ([8])**

A non-empty subset $B$ of the variables of the formula $F$ ($B \subseteq \text{var}(F)$) is a **strong backdoor** w.r.t. the subsolver $C$ for $F$ iff for all interpretations $J : B \rightarrow \{\top, \bot\}$, $C$ returns a satisfying assignment or concludes unsatisfiability of $F|_J$.

$F − V$: replaces in each clause of $F$ the occurrences of $x$ and $\neg x$ with $\bot$ for each variable $x \in V$ (with $V \subseteq \text{var}(F)$) and simplifies the clause.

**Definition: Deletion $C$-backdoor ([4])**

A non-empty subset $B$ of the variables of the formula $F$ ($B \subseteq \text{var}(F)$) is a **deletion backdoor** w.r.t. a class $C$ for $F$ iff $F − B \in C$. 
$F_J$: reduct of $F$ w.r.t. the (partial) interpretation $J$; it is obtained by replacing each variable $v$ in $F$ with $J(v)$ and simplifying.

**Definition: Strong $C$-Backdoor ([8])**

A non-empty subset $B$ of the variables of the formula $F$ ($B \subseteq \text{var}(F)$) is a *strong backdoor* w.r.t. the subsolver $C$ for $F$ iff for all interpretations $J : B \rightarrow \{\top, \bot\}$, $C$ returns a satisfying assignment or concludes unsatisfiability of $F_J$.

$F - V$: replaces in each clause of $F$ the occurrences of $x$ and $\neg x$ with $\bot$ for each variable $x \in V$ (with $V \subseteq \text{var}(F)$) and simplifies the clause.

**Definition: Deletion $C$-backdoor ([4])**

A non-empty subset $B$ of the variables of the formula $F$ ($B \subseteq \text{var}(F)$) is a *deletion backdoor* w.r.t. a class $C$ for $F$ iff $F - B \in C$. 
For some classes, like Horn, deletion backdoor and strong backdoor are equivalent. ([2])

Example
This is good! To verify whether $B$ is a deletion $C$-backdoor we just need to check whether $F - B$ is in $C$ and not whether all the $2^{|B|}$ reducts of $F$ are.
Kernelization

Given a parameterized problem \((x, k)\), a kernelization is a polynomial time preprocessing technique that either:
- returns a new equisatisfiable instance \((x', k')\) (with \(|x'| \leq g(k)\) and \(k' \leq k\)),
- rejects the instance as unsatisfiable.

The high degree kernelization is simple but powerful:

**HdK** A vertex of degree \(> k\) must be in any cover of size \(\leq k\).

By applying some other pre-processing to our graph, we can apply the following result:

**Property ([1])**

If \(G\) is a graph with a vertex cover of size \(k\) and there is no vertex of \(G\) with degree \(> k\) or degree \(< 3\), then \(|V| \leq \frac{k^2}{3} + k\).