Backdoors for SAT

Marco Gario

EMCL / TUD

June 27, 2011

Backdoors in a nutshell

What?

Given a combinatorial problem, we call *backdoor variables set* (backdoors set) a set of variables that, once decided, make the rest of the problem "easy" to solve.

Why?

Backdoors have been introduced by Williams et al. ([12]) to try to explain the good performances of modern SAT solvers.

Content

Outline

Notation (1/2)

We refresh the usual notation:

- F : Formula (possibly in CNF)
- $var(F)$: the set of variables appearing in F.
- $v, \overline{v}$: a variable v or its negation $\overline{v}$
- J : (partial) interpretation. (Partial) mapping from $var(F)$ to the boolean values $\{\top, \bot\}$. We represent an interpretation compactly by listing the literals in it. Eg. $J = \{x_1, \overline{x_3}\}$
- $F|_J$: reduct of F w.r.t. the (partial) interpretation J; it is obtained by replacing each variable v in F with $J(v)$.

Notation (2/2)

Definition

Given a CNF formula F and a set of variables $V' \subseteq var(F)$ we denote with $F - V'$ the formula obtained from F by removing all the occurrences of the variables in V' from F.

Example

Given a CNF formula F and $V = \{e\}$ we obtain:

$$\begin{aligned} F - \{e\} = & a \wedge \neg b \wedge (d \vee \cancel{e}) \wedge (\cancel{\neg e} \vee \neg d) = \\ = & a \wedge \neg b \wedge d \wedge \neg d \end{aligned}$$

Outline

Class

We call *class* a set of formulas that share some property.
Well-known classes of SAT problems are:

- 2SAT:For F in CNF: $F \in 2SAT$ iff each clause of F has at most two literals
- Horn:For F in CNF: $F \in Horn$ iff each clause of F has at most one positive literal
- Renamable Horn (RHorn)
- Unit Propagation and Pure Literal (UP+PL)

Class

We call *class* a set of formulas that share some property.
Well-known classes of SAT problems are:

- 2SAT:For F in CNF: $F \in 2SAT$ iff each clause of F has at most two literals
- Horn:For F in CNF: $F \in Horn$ iff each clause of F has at most one positive literal
- Renamable Horn (RHorn)
- Unit Propagation and Pure Literal (UP+PL)

Renamable Horn (RHorn)

Definition: Variable flipping

We call a *variable flipping* for the variable $x \in var(F)$, the substitution of all occurences of x in F with $\neg x$ and, similarly, of all $\neg x$ with x.

- For F CNF formula: $F \in RHorn$ iff there exists a set of variables that, once flipped, make the formula in Horn.

Example

$$F_b = (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_2)$$

$F_b \notin Horn$ but $F_b \in RHorn$ because of the flipping $\{x_1\}$:

$$(\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2)$$

Unit Propagation and Pure Literal (UP+PL)

- For F CNF formula: $F \in UP + PL$ iff it can be solved by applying only unit propagation and pure literal elimination to F

Example

$$F_c = x_1 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (x_3 \vee x_4)$$

Unit Propagation and Pure Literal (UP+PL)

- For F CNF formula: $F \in UP + PL$ iff it can be solved by applying only unit propagation and pure literal elimination to F

Example

$$
\begin{aligned}
F_c =& x_1 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (x_3 \vee x_4) \\
F_c|_{\{x_4\}} =& x_1 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge \cancel{(x_3 \vee x_4)}
\end{aligned}
$$

Unit Propagation and Pure Literal (UP+PL)

- For F CNF formula: $F \in UP + PL$ iff it can be solved by applying only unit propagation and pure literal elimination to F

Example

$$\begin{aligned}
F_c =& x_1 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (x_3 \vee x_4) \\
F_c|_{\{x_4\}} =& x_1 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge \cancel{(x_3 \vee x_4)} \\
F_c|_{\{x_4, x_1\}} =& \cancel{x_1} \wedge (\cancel{\neg x_1} \vee x_3) \wedge \cancel{(\neg x_2 \vee x_1)} \wedge (\neg x_3 \vee x_2) \\
=& x_3 \wedge (\neg x_3 \vee x_2)
\end{aligned}$$

Unit Propagation and Pure Literal (UP+PL)

- For F CNF formula: $F \in UP + PL$ iff it can be solved by applying only unit propagation and pure literal elimination to F

Example

$$
\begin{aligned}
F_c &= x_1 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (x_3 \vee x_4) \\
F_c|_{\{x_4\}} &= x_1 \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge \cancel{(x_3 \vee x_4)} \\
F_c|_{\{x_4, x_1\}} &= \cancel{x_1} \wedge (\cancel{\neg x_1} \vee x_3) \wedge \cancel{(\neg x_2 \vee x_1)} \wedge (\neg x_3 \vee x_2) \\
&= x_3 \wedge (\neg x_3 \vee x_2) \\
F_c|_{\{x_4, x_1, x_3\}} &= \cancel{x_3} \wedge (\cancel{\neg x_3} \vee x_2) = x_2 \\
F_c|_{\{x_4, x_1, x_3, x_2\}} &= \top
\end{aligned}
$$

Class properties (1/2)

Definition: Clause Induced

A class $\mathcal{C}$ is said to be *clause induced* whenever a formula belongs to the class iff each of its clauses (viewed as a formula) belongs to the class; i.e. $F \in \mathcal{C} \leftrightarrow \forall G_i \in F . G_i \in \mathcal{C}$

A weaker property is being closed under clause removal:

Definition: Closed under clause removal

A class $\mathcal{C}$ is *closed under clause removal* if for all formulas in the class, it holds that each subset of the clauses (when treated as a formula) belongs to the class; i.e. $\forall F \in \mathcal{C}, \forall F' \subseteq F$ it holds that $F' \in \mathcal{C}$

Class properties (2/2)

Example

Horn and 2SAT are both *closed under clause removal* and *clause induced*:

$$F_a = (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2)$$

$F_a \in Horn$ and $\forall F' \subseteq F \in Horn$.
$F_a \in 2SAT$ and $\forall F' \subseteq F \in 2SAT$.

$G_1 = F_a$

$G_2 = (\neg x_1 \vee \neg x_3)$

$G = G_1 \wedge G_2 = (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_1 \vee \neg x_3)$

$G_2 \in Horn$, $G \in Horn$. $G_2 \in 2SAT$, $G \in 2SAT$.

Class properties (2/2)

Example

Horn and 2SAT are both *closed under clause removal* and *clause induced*:

$$F_a = (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2)$$

$F_a \in Horn$ and $\forall F' \subseteq F \in Horn$.
$F_a \in 2SAT$ and $\forall F' \subseteq F \in 2SAT$.

$G_1 = F_a$
$G_2 = (\neg x_1 \vee \neg x_3)$
$G = G_1 \wedge G_2 = (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_1 \vee \neg x_3)$

$G_2 \in Horn$, $G \in Horn$. $G_2 \in 2SAT$, $G \in 2SAT$.

Class properties (2/2)

Example

RHorn is only *closed under clause removal* but not *clause induced*:

$$F_b = (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_2)$$

$F_b \in RHorn$ for the flipping $\{x_1\}$ and $\forall F' \subseteq F \in RHorn$ for the same flipping.

$G_1 = F_b$
$G_2 = (\neg x_1 \vee x_3)$
$G = G_1 \wedge G_2 = (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_1 \vee x_3)$

$G \notin RHorn.$

Class properties (2/2)

Example

RHorn is only *closed under clause removal* but not *clause induced*:

$$F_b = (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_2)$$

$F_b \in RHorn$ for the flipping $\{x_1\}$ and $\forall F' \subseteq F \in RHorn$ for the same flipping.

$$G_1 = F_b$$
$$G_2 = (\neg x_1 \vee x_3)$$
$$G = G_1 \wedge G_2 = (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_1) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_1 \vee x_3)$$

$G \notin RHorn$.

Notational Disclaimer

Disclaimer

Some authors (eg. Szeider, Nishimura, Samer and Kottler) use "clause induced" to indicate what we call "closed under clause removal."

Subsolvers (1/2)

We are interested in classes for which there is a “good” solving algorithm:

Definition: Subsolver [12]

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 v of the variable x C determines $F|_{\{x \mapsto v\}}$

Subsolvers (1/2)

We are interested in classes for which there is a “good” solving algorithm:

Definition: Subsolver [12]

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), **Weakening this axiom we obtain heuristic backdoors [9]**
- 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 v of the variable x C determines $F|_{\{x \mapsto v\}}$

Subsolvers (1/2)

We are interested in classes for which there is a “good” solving algorithm:

Definition: Subsolver [12]

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, **Removing this axiom we obtain pseudo backdoors [9]**

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 v of the variable x C determines $F|_{\{x \mapsto v\}}$

Subsolvers (2/2)

There exists a subsolver for: 2SAT, Horn, RHorn and UP+PL

In the following we do not distinguish between subsolver C and related class $\mathcal{C}$.

Subsolvers (2/2)

There exists a subsolver for: 2SAT, Horn, RHorn and UP+PL

In the following we do not distinguish between subsolver C and related class $\mathcal{C}$.

Outline

Strong/Weak Backdoors (1/3)

Definition: Strong C-Backdoor

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

If a formula F is satisfiable, we can define a simpler type of backdoor:

Definition: Weak C-Backdoor

A non-empty subset B of the variables of the formula F ($B \subseteq var(F)$) is a *weak backdoor* w.r.t. the subsolver C for F iff **there exists** a interpretation $J : B \to \{\top, \bot\}$ such that C returns a satisfying assignment of $F|_J$.

Strong/Weak Backdoors (2/3)

Example

Lets consider the satisfiable formula F_0:

$$F_0 = (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1) \wedge (\neg x_1 \vee x_6) \wedge (\neg x_1 \vee \neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6 \vee x_2)$$

$B = \{x_1, x_2\}$ is a strong 2SAT-backdoor and therefore, since F_0 is satisfiable, it is also a weak 2SAT-backdoor:

$J_0 = \{x_1, x_2\}$, $J_1 = \{x_1, \bar{x_2}\}$, $J_2 = \{\bar{x_1}, x_2\}$ and $J_3 = \{\bar{x_1}, \bar{x_2}\}$.

Strong/Weak Backdoors (2/3)

Example

Lets consider the satisfiable formula F_0:

$$F_0 = (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1) \wedge (\neg x_1 \vee x_6) \wedge \\ (\neg x_1 \vee \neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6 \vee x_2)$$

$B = \{x_1, x_2\}$ is a strong 2SAT-backdoor and therefore, since F_0 is satisfiable, it is also a weak 2SAT-backdoor:
$J_0 = \{x_1, x_2\}$, $J_1 = \{x_1, \bar{x_2}\}$, $J_2 = \{\bar{x_1}, x_2\}$ and $J_3 = \{\bar{x_1}, \bar{x_2}\}$.

Strong/Weak Backdoors (2/3)

Example

Lets consider the satisfiable formula F_0:

$$F_0 = (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1) \wedge (\neg x_1 \vee x_6) \wedge$$
$$(\neg x_1 \vee \neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6 \vee x_2)$$

$B = \{x_1, x_2\}$ is a strong 2SAT-backdoor and therefore, since F_0 is satisfiable, it is also a weak 2SAT-backdoor:
$J_0 = \{x_1, x_2\}$, $J_1 = \{x_1, \bar{x_2}\}$, $J_2 = \{\bar{x_1}, x_2\}$ and $J_3 = \{\bar{x_1}, \bar{x_2}\}$.

$$F_0|_{J_0} = x_3 \wedge x_6 \wedge (\neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4)$$

Strong/Weak Backdoors (2/3)

Example

Lets consider the satisfiable formula F_0:

$$F_0 = (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1) \wedge (\neg x_1 \vee x_6) \wedge (\neg x_1 \vee \neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6 \vee x_2)$$

$B = \{x_1, x_2\}$ is a strong 2SAT-backdoor and therefore, since F_0 is satisfiable, it is also a weak 2SAT-backdoor:
$J_0 = \{x_1, x_2\}$, $J_1 = \{x_1, \bar{x_2}\}$, $J_2 = \{\bar{x_1}, x_2\}$ and $J_3 = \{\bar{x_1}, \bar{x_2}\}$.

$$F_0|_{J_1} = x_6 \wedge (\neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6)$$

Strong/Weak Backdoors (2/3)

Example

Lets consider the satisfiable formula F_0:

$$F_0 = (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1) \wedge (\neg x_1 \vee x_6) \wedge (\neg x_1 \vee \neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6 \vee x_2)$$

$B = \{x_1, x_2\}$ is a strong 2SAT-backdoor and therefore, since F_0 is satisfiable, it is also a weak 2SAT-backdoor:
$J_0 = \{x_1, x_2\}$, $J_1 = \{x_1, \bar{x_2}\}$, $J_2 = \{\bar{x_1}, x_2\}$ and $J_3 = \{\bar{x_1}, \bar{x_2}\}$.

$$F_0|_{J_2} = x_3 \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_5 \vee x_4)$$

Strong/Weak Backdoors (2/3)

Example

Lets consider the satisfiable formula F_0:

$$F_0 = (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1) \wedge (\neg x_1 \vee x_6) \wedge (\neg x_1 \vee \neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6 \vee x_2)$$

$B = \{x_1, x_2\}$ is a strong 2SAT-backdoor and therefore, since F_0 is satisfiable, it is also a weak 2SAT-backdoor:
$J_0 = \{x_1, x_2\}$, $J_1 = \{x_1, \bar{x_2}\}$, $J_2 = \{\bar{x_1}, x_2\}$ and $J_3 = \{\bar{x_1}, \bar{x_2}\}$.

$$F_0|_{J_3} = (\neg x_3 \vee \neg x_4) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6)$$

Strong/Weak Backdoors (3/3)

For a given formula F we define:

Definition: Minimal backdoor

A strong (resp. weak) C-backdoor B is called *minimal* iff there is no proper subset of B that is a strong (weak) $\mathcal{C}$-backdoor, i.e. $\forall B' \subset B$, B' is not a strong (weak) $\mathcal{C}$-backdoor.

Eg. The set of all variables of a SAT problem is a backdoor (for any subsolver) but, most likely, it is not minimal.

Definition: Smallest backdoor

A strong (resp. weak) C-backdoor B is called *smallest* iff it is minimal and $|B| \leq |B'|$ for any minimal C-backdoor B'

Note: There can be more than one smallest C-backdoor for the same formula F!

Strong/Weak Backdoors (3/3)

For a given formula F we define:

Definition: Minimal backdoor

A strong (resp. weak) C-backdoor B is called *minimal* iff there is no proper subset of B that is a strong (weak) $\mathcal{C}$-backdoor, i.e. $\forall B' \subset B$, B' is not a strong (weak) $\mathcal{C}$-backdoor.

Eg. The set of all variables of a SAT problem is a backdoor (for any subsolver) but, most likely, it is not minimal.

Definition: Smallest backdoor

A strong (resp. weak) C-backdoor B is called *smallest* iff it is minimal and $|B| \leq |B'|$ for any minimal C-backdoor B'

Note: There can be more than one smallest C-backdoor for the same formula F!

Deletion (1/2)

Definition: Deletion C-backdoor [6]

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

Example

$F_0 \notin 2SAT$:

$$F_0 = (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_1) \wedge (\neg x_1 \vee x_6) \wedge (\neg x_1 \vee \neg x_6 \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg x_6 \vee x_2)$$

but $F_0 - \{x_1, x_2\} \in 2SAT$. Recall from a previous example that $B = \{x_1, x_2\}$ is a strong 2SAT-backdoor for F_0

Deletion (2/2)

The following properties make deletion backdoors interesting:

Property ([6])

If the class $\mathcal{C}$ is closed under clause removal then every deletion C-backdoor is also a strong C-backdoor (deletion $\rightarrow$ strong)

Property ([2])

If the class $\mathcal{C}$ is clause induced (eg. 2SAT or Horn) then strong C-backdoor and deletion C-backdoor are equivalent (deletion $\leftrightarrow$ strong).

Given F and $|B| = k$ we need to perform only 1 test ($F - B \in \mathcal{C}$) instead of 2^k!

Outline

Learning-sensitive (1/4)

Definition: Search tree exploration

Given a formula F, we call *search tree exploration* an ordered list of literals $(l_1, .., l_n)$ such that $l_i \in \{v_i, \bar{v}_i\}$ with $v_i \in var(F)$.

Example

Lets consider the search tree exploration $(x_1, x_2, \bar{x_3}, \bar{x_1}, x_3)$ for the following formula:

$$F = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1)$$

the search algorithm traversed the search space as follows:

Learning-sensitive (1/4)

Definition: Search tree exploration

Given a formula F, we call *search tree exploration* an ordered list of literals $(l_1, .., l_n)$ such that $l_i \in \{v_i, \bar{v}_i\}$ with $v_i \in var(F)$.

Example

Lets consider the search tree exploration $(x_1, x_2, \bar{x}_3, \bar{x}_1, x_3)$ for the following formula:

$$F = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1)$$

the search algorithm traversed the search space as follows:

$$F|_{\{x_1\}} = x_2 \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$$

Learning-sensitive (1/4)

Definition: Search tree exploration

Given a formula F, we call *search tree exploration* an ordered list of literals $(l_1, .., l_n)$ such that $l_i \in \{v_i, \bar{v}_i\}$ with $v_i \in var(F)$.

Example

Lets consider the search tree exploration $(x_1, x_2, \bar{x}_3, \bar{x}_1, x_3)$ for the following formula:

$$F = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1)$$

the search algorithm traversed the search space as follows:

$$F|_{\{x_1, x_2\}} = (x_3) \wedge (\neg x_3)$$

Learning-sensitive (1/4)

Definition: Search tree exploration

Given a formula F, we call *search tree exploration* an ordered list of literals $(l_1, .., l_n)$ such that $l_i \in \{v_i, \bar{v_i}\}$ with $v_i \in var(F)$.

Example

Lets consider the search tree exploration $(x_1, x_2, \bar{x_3}, \bar{x_1}, x_3)$ for the following formula:

$$F = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1)$$

the search algorithm traversed the search space as follows:

$$F|_{\{x_1, x_2, \bar{x_3}\}} = \perp$$

Learning-sensitive (1/4)

Definition: Search tree exploration

Given a formula F, we call *search tree exploration* an ordered list of literals $(l_1, .., l_n)$ such that $l_i \in \{v_i, \bar{v_i}\}$ with $v_i \in var(F)$.

Example

Lets consider the search tree exploration $(x_1, x_2, \bar{x_3}, \bar{x_1}, x_3)$ for the following formula:

$$F = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1)$$

the search algorithm traversed the search space as follows:

$$F|_{\{\bar{x_1}\}} = (\neg x_2 \vee x_3)$$

Learning-sensitive (1/4)

Definition: Search tree exploration

Given a formula F, we call *search tree exploration* an ordered list of literals $(l_1, .., l_n)$ such that $l_i \in \{v_i, \bar{v_i}\}$ with $v_i \in var(F)$.

Example

Lets consider the search tree exploration $(x_1, x_2, \bar{x_3}, \bar{x_1}, x_3)$ for the following formula:

$$F = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1)$$

the search algorithm traversed the search space as follows:

$$F|_{\{\bar{x_1}, x_3\}} = \top$$

Learning-sensitive (2/4)

How does clause learning influence backdoors?

Definition: Learning-sensitive backdoors [3]

A non-empty subset of variables B of the formula F is a *learning-sensitive* C-backdoor for F iff there exists a search tree exploration such that a clause learning SAT solver branching only on the variables in B, in this order and with C as subsolver at every leaf of the search tree, either finds a satisfying assignment or proves F unsatisfiable.

Learning-sensitive (3/4)

Example

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

Claim: $B = \{x\}$ is a learning-sensitive UP+PL-backdoor for F_1 for the search tree exploration $(\bar{x}, p_1, p_2, q, a, b, x, \bar{q}, r, a, b)$.

Learning-sensitive (3/4)

Example

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

Claim: $B = \{x\}$ is a learning-sensitive UP+PL-backdoor for F_1 for the search tree exploration $(\bar{x}, p_1, p_2, q, a, b, x, \bar{q}, r, a, b)$.

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

Learning-sensitive (3/4)

Example

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

Claim: $B = \{x\}$ is a learning-sensitive UP+PL-backdoor for F_1 for the search tree exploration $(\bar{x}, p_1, p_2, q, a, b, x, \bar{q}, r, a, b)$.

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

Learning-sensitive (3/4)

Example

$$F_1 = (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee q) \wedge$$
$$(\neg q \vee a) \wedge (\neg q \vee \neg a \vee b) \wedge (\neg q \vee \neg a \vee \neg b) \wedge$$
$$(\neg x \vee q \vee r) \wedge (\neg r \vee a) \wedge (\neg r \vee \neg a \vee b) \wedge (\neg r \vee \neg a \vee \neg b)$$

Claim: $B = \{x\}$ is a learning-sensitive UP+PL-backdoor for F_1 for the search tree exploration $(\bar{x}, p_1, p_2, q, a, b, x, \bar{q}, r, a, b)$.

$$F_1|_{\{\bar{x}, p_1, p_2, q, a\}} = b \wedge \neg b \wedge (\neg r \vee b) \wedge (\neg r \vee \neg b)$$

Learning-sensitive (3/4)

Example

$$F_1{}' = (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg p_1 \vee \neg p_2 \vee q) \wedge \\ (\neg q \vee a) \wedge (\neg q \vee \neg a \vee b) \wedge (\neg q \vee \neg a \vee \neg b) \wedge \\ (\neg x \vee q \vee r) \wedge (\neg r \vee a) \wedge (\neg r \vee \neg a \vee b) \wedge (\neg r \vee \neg a \vee \neg b) \\ \wedge \neg q$$

Claim: $B = \{x\}$ is a learning-sensitive UP+PL-backdoor for F_1 for the search tree exploration $(\bar{x}, p_1, p_2, q, a, b, x, \bar{q}, r, a, b)$.
Conflict: 1-UIP learning scheme gives us the clause $\neg q$

Learning-sensitive (3/4)

Example

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

Claim: $B = \{x\}$ is a learning-sensitive UP+PL-backdoor for F_1 for the search tree exploration $(\bar{x}, p_1, p_2, q, a, b, x, \bar{q}, r, a, b)$.

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

Learning-sensitive (3/4)

Example

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

Claim: $B = \{x\}$ is a learning-sensitive UP+PL-backdoor for F_1 for the search tree exploration $(\bar{x}, p_1, p_2, q, a, b, x, \bar{q}, r, a, b)$.

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

Learning-sensitive (3/4)

Example

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

The tree search exploration is important, since $B = \{x\}$ is not a UP+PL-backdoor if we consider x before $\bar{x}$:

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

Learning-sensitive (3/4)

Example

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

$$\begin{aligned} F_1|_{\{x, \bar{p}_1\}} = &(\neg q \vee a) \wedge \\ &(\neg q \vee \neg a \vee b) \wedge (\neg q \vee \neg a \vee \neg b) \wedge \\ &(\vee q \vee r) \wedge (\neg r \vee a) \wedge (\neg r \vee \neg a \vee b) \wedge (\neg r \vee \neg a \vee \neg b) \end{aligned}$$

at this point UP+PL is not enough to decide the formula.

Learning-sensitive (4/4)

Why are learning-sensitive backdoors interesting?

Property ([3])

There are SAT instances for which the smallest learning-sensitive UP-backdoors are smaller than the smallest strong UP-backdoor (even exponentially if the instance is unsat)

Property ([3])

There are unsatisfiable SAT instances for which one value-ordering of the variables can lead to exponentially smaller learning-sensitive UP-backdoor than a different value ordering

Trees (1/3)

Definition: Decision Tree

A binary *decision tree* is a rooted binary tree T, such that every node in T is either a leaf or has exactly 2 children. The nodes of T, except for the root, are labeled with literals s.t. the following conditions are satisfied:

- two nodes v_i and v_j with the same father are labeled with complementary literals x and $\bar{x}$;
- the labels of the node on a path from the root to a leaf do not contain the same literal twice nor a complementary pair of literals.

We call J_v the partial interpretation expressed by the path that links the root to the node v, and $var(T)$ the set of variables appearing in T.

Trees (2/3)

Definition: Backdoor Tree

A binary decision tree T (with $var(T) \subseteq var(F)$) is a *C-backdoor tree* of F if $F|_{J_v} \in \mathcal{C}$ for every leaf v of T.

Backdoor trees are interesting because of the following property:

Property ([10])

If B is a smallest strong C-backdoor for F and T is a smallest C-backdoor tree of F, then:

$$|B| + 1 \leq |T| \leq 2^{|B|}$$

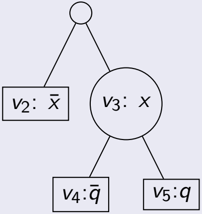
where $|T|$ is the number of leaves of T.

Trees (3/3)

Example

$$F_1 = (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg x \vee q \vee r) \wedge H$$

$B_1 = \{x, q\}$ is a strong Horn-backdoor for F_1, but neither $\{x\}$ nor $\{q\}$ are. The Horn-backdoor tree for F_1 looks like:

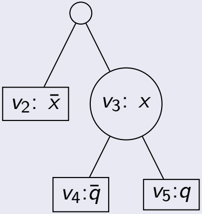


Trees (3/3)

Example

$$F_1 = (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg x \vee q \vee r) \wedge H$$

$B_1 = \{x, q\}$ is a strong Horn-backdoor for F_1, but neither $\{x\}$ nor $\{q\}$ are. The Horn-backdoor tree for F_1 looks like:



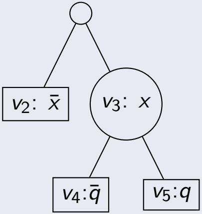
In particular, we note that there are 3 leaf nodes (v2,v4,v5) and therefore 3 partial interpretations that lead to a Horn formula: $J_{v2} = \{\bar{x}\}$, $J_{v4} = \{x, \bar{q}\}$ and $J_{v5} = \{x, q\}$.

Trees (3/3)

Example

$$F_1 = (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg x \vee q \vee r) \wedge H$$

$B_1 = \{x, q\}$ is a strong Horn-backdoor for F_1, but neither $\{x\}$ nor $\{q\}$ are. The Horn-backdoor tree for F_1 looks like:



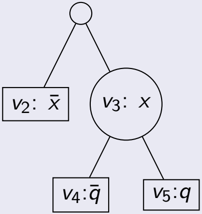
Without using the concept of backdoor trees, we would end up testing more assignments, in this case $2^{|B|} = 4$.

Trees (3/3)

Example

$$F_1 = (x \vee p_1) \wedge (x \vee p_2) \wedge (\neg x \vee q \vee r) \wedge H$$

$B_1 = \{x, q\}$ is a strong Horn-backdoor for F_1, but neither $\{x\}$ nor $\{q\}$ are. The Horn-backdoor tree for F_1 looks like:



Note that variable order is important for the size of the tree, a bigger backdoor tree for F_1 is obtained by considering q before x and has 4 leaf nodes.

Outline

Complexity Intro

- We looked into classes/subsolvers that run in P,

$\Rightarrow$ Given a backdoor set for a formula, we can decide satisfiability in polynomial time

- *SAT* is NP-complete and assuming $P \neq NP \Rightarrow$ finding the backdoor set is NP-hard

Complexity Intro

- We looked into classes/subsolvers that run in P,

$\Rightarrow$ Given a backdoor set for a formula, we can decide satisfiability in polynomial time

- SAT is NP-complete and assuming $P \neq NP \Rightarrow$ finding the backdoor set is NP-hard

Parameterized Complexity in one Slide

- Parameterized complexity is an interesting (but not really known) field in which we "cheat" when measuring complexity
- Classical complexity: Worst-case runtime in the size n of the input
- Parameterized complexity: Worst-case runtime both in the size n of the input AND of a parameter k
- SAT: $O(2^n)$
- p-SAT: SAT parameterized by the number of variables $\Rightarrow O(2^k n^c)$, for c constant
- Algorithms that are function only of the parameter k are called *fixed parameter tractable* (FPT) : $O(f(k)n^c)$
- Observation: p-SAT is in P by considering classical complexity!
- Question: Can we find a parameter for backdoor detection to obtain a polynomial (ie. FPT) running time?

Parameterized Complexity in one Slide

- Parameterized complexity is an interesting (but not really known) field in which we "cheat" when measuring complexity
- Classical complexity: Worst-case runtime in the size n of the input
- Parameterized complexity: Worst-case runtime both in the size n of the input AND of a parameter k
- SAT: $O(2^n)$
- p-SAT: SAT parameterized by the number of variables $\Rightarrow O(2^k n^c)$, for c constant
- Algorithms that are function only of the parameter k are called *fixed parameter tractable* (FPT) : $O(f(k)n^c)$
- Observation: p-SAT is in P by considering classical complexity!
- Question: Can we find a parameter for backdoor detection to obtain a polynomial (ie. FPT) running time?

Parameterized Complexity in one Slide

- Parameterized complexity is an interesting (but not really known) field in which we “cheat” when measuring complexity
- Classical complexity: Worst-case runtime in the size n of the input
- Parameterized complexity: Worst-case runtime both in the size n of the input AND of a parameter k
- SAT: $O(2^n)$
- p-SAT: SAT parameterized by the number of variables $\Rightarrow O(2^k n^c)$, for c constant
- Algorithms that are function only of the parameter k are called *fixed parameter tractable* (FPT) : $O(f(k)n^c)$
- Observation: p-SAT is in P by considering classical complexity!
- Question: Can we find a parameter for backdoor detection to obtain a polynomial (ie. FPT) running time?

Parameterized Complexity in one Slide

- Parameterized complexity is an interesting (but not really known) field in which we "cheat" when measuring complexity
- Classical complexity: Worst-case runtime in the size n of the input
- Parameterized complexity: Worst-case runtime both in the size n of the input AND of a parameter k
- SAT: $O(2^n)$
- p-SAT: SAT parameterized by the number of variables $\Rightarrow O(2^k n^c)$, for c constant
- Algorithms that are function only of the parameter k are called *fixed parameter tractable* (FPT) : $O(f(k)n^c)$
- Observation: p-SAT is in P by considering classical complexity!
- Question: Can we find a parameter for backdoor detection to obtain a polynomial (ie. FPT) running time?

Parameterized backdoor detection

Definition: {weak, strong, deletion, tree} C-backdoor detection

Input: A CNF formula F ;

Parameter: An integer $k \geq 0$, size of the backdoor;

Question: Does F have a {weak, strong, deletion, tree} C-backdoor of size at most k?

This problem can be solved in $O(n^k)$ for $k << n$, can we do better?

Class	Weak	Strong	Deletion	Tree
2SAT,Horn	W[2]-complete[1]	FPT[1]	FPT[1]	FPT[2]
RHorn		W[1]-hard[3]	FPT[4]	
UP,PL,UP+PL	W[P]-complete[5]	W[P]-complete[5]		

$1 = [7], 2 = [10], 3 = [1], 4 = [8], 5 = [11]$

Parameterized backdoor detection

Definition: {weak, strong, deletion, tree} C-backdoor detection

Input: A CNF formula F ;

Parameter: An integer $k \geq 0$, size of the backdoor;

Question: Does F have a {weak, strong, deletion, tree} C-backdoor of size at most k?

This problem can be solved in $O(n^k)$ for $k << n$, can we do better?

Class	Weak	Strong	Deletion	Tree
2SAT,Horn	W[2]-complete[1]	FPT[1]	FPT[1]	FPT[2]
RHorn		W[1]-hard[3]	FPT[4]	
UP,PL,UP+PL	W[P]-complete[5]	W[P]-complete[5]		

1 = [7],2 = [10],3 = [1], 4 = [8],5 = [11]

What does this mean?

- The problems that are FPT, can be solved efficiently if the formula has a (small) backdoor of that size.
- Eg. For Horn and 2SAT we have agorithms for strong/deletion that run in $O(2^k n)$ and $O(3^k n)$ respectively([7])
- There are several techniques that have been developed for FPT problems, that might allow us to improve the above result: kernelization, iterative compression etc.

What does this mean?

- The problems that are FPT, can be solved efficiently if the formula has a (small) backdoor of that size.
- Eg. For Horn and 2SAT we have agorithms for strong/deletion that run in $O(2^k n)$ and $O(3^k n)$ respectively([7])
- There are several techniques that have been developed for FPT problems, that might allow us to improve the above result: kernelization, iterative compression etc.

Outline

1. Introduction
   - Notation
   - Classes and Subsolvers
2. Backdoors
   - Strong, Weak, Deletion
   - Extensions: Learning-Sensitive, Trees
3. General Results
   - Complexity Highlights
   - Experimental results
4. Conclusions

Problems in experimental data

Most of the experimental work done on backdoors does not consider the FPT complexity. Mainly two approaches are used:

- Local Search: Get a variable set, "shuffle" it until it is a backdoor, remove unneeded elements (minimize) $\Rightarrow$ cannot guarantee to find smallest backdoors, provides only an idea on the upperbound of the backdoor size.
- Complete: test every n^k subset of k variables $\Rightarrow$ easy to find smallest backdoor, unfeasible for almost any value of k

Most of the experimental work considers different classes $\mathcal{C}$: use a SAT solver as subsolver (eg. Satz) study Satz-backdoors $\Rightarrow$ hard to put results together

Problems in experimental data

Most of the experimental work done on backdoors does not consider the FPT complexity. Mainly two approaches are used:

- Local Search: Get a variable set, "shuffle" it until it is a backdoor, remove unneeded elements (minimize) $\Rightarrow$ cannot guarantee to find smallest backdoors, provides only an idea on the upperbound of the backdoor size.
- Complete: test every n^k subset of k variables $\Rightarrow$ easy to find smallest backdoor, unfeasible for almost any value of k

Most of the experimental work considers different classes $\mathcal{C}$: use a SAT solver as subsolver (eg. Satz) study Satz-backdoors $\Rightarrow$ hard to put results together

Problems in experimental data

Most of the experimental work done on backdoors does not consider the FPT complexity. Mainly two approaches are used:

- Local Search: Get a variable set, “shuffle” it until it is a backdoor, remove unneeded elements (minimize) $\Rightarrow$ cannot guarantee to find smallest backdoors, provides only an idea on the upperbound of the backdoor size.
- Complete: test every n^k subset of k variables $\Rightarrow$ easy to find smallest backdoor, unfeasible for almost any value of k

Most of the experimental work considers different classes $\mathcal{C}$: use a SAT solver as subsolver (eg. Satz) study Satz-backdoors $\Rightarrow$ hard to put results together

Problems in experimental data

Most of the experimental work done on backdoors does not consider the FPT complexity. Mainly two approaches are used:

- Local Search: Get a variable set, "shuffle" it until it is a backdoor, remove unneeded elements (minimize) $\Rightarrow$ cannot guarantee to find smallest backdoors, provides only an idea on the upperbound of the backdoor size.
- Complete: test every n^k subset of k variables $\Rightarrow$ easy to find smallest backdoor, unfeasible for almost any value of k

Most of the experimental work considers different classes $\mathcal{C}$: use a SAT solver as subsolver (eg. Satz) study Satz-backdoors $\Rightarrow$ hard to put results together

Experimental overview

Nevertheless we can try to answer the question: Are backdoor sets small w.r.t. the number of variables?

- Dilkina et al. ([4]) provide a nice comparison among different classes of backdoors establishing the following "order": Horn ($9-67\%$), del RHorn ($2-66\%$),UP ($0.15-12\%$), UP+PL ($0.13-1.4\%$) and Satz ($0-0.6\%$);
- Samer and Szeider ([10]) confirm that backdoor trees are smaller than strong backdoors (for Horn and RHorn)

Experimental overview

? Random instances: Gregory et al. ([5]) show how weak UP+PL-backdoors are usually bigger in random instances (10%) than in structured SAT ($2 - 5\%$). No information is available for other classes.

? Preprocessing: The results form Dilkina et al. ([4]) are somehow inconclusive. In some cases pre-processing is "bad", in other is not relevant. Moreover they are not considering smallest backdoor but upperbounds.

Conclusions

- Backdoors allow to solve efficiently a SAT problem
- There are different types and classes of backdoors
- Finding backdoors is hard, but not-that-hard for some classes/types (FPT)
- Experimental results are hard generalize and compare. Moreover, they require an awful amount of time to be repeated.

What is next?

- Define more polynomial time classes, and study their detection complexity
- Study the concept of backdoor by removing the "efficiency" constraint
- ⋆ Apply FPT techniques to FPT detection problems to speed-up the search (eg. kernelization, parallelization and iterative compression)
- ⋆ Try to identify domains or properties of SAT instances for which C-backdoors are small, and C-backdoor detection can be solved in FPT.
- Devise heuristics that "guess" backdoors incredible well!
- Apply the backdoor idea to other domains: eg. QBF, ASP, #SAT have been covered by Szeider.

References I

A. Biere, M. Heule, H. van Maaren, and T. Walsh.
Handbook of Satisfiability.
IOS Press, 2009.

Y. Crama, O. Ekin, and P.L. Hammer.
Variable and term removal from Boolean formulae.
Discrete Applied Mathematics, 75(3):217–230, 1997.

Bistra Dilkina, C. Gomes, and Ashish Sabharwal.
Backdoors in the Context of Learning.
Theory and Applications of Satisfiability Testing-SAT 2009, pages 73–79, 2009.

References II

Bistra Dilkina, C.P. Gomes, and Ashish Sabharwal.
Tradeoffs in backdoors: Inconsistency detection, dynamic simplification, and preprocessing.
In *ISAIM-08: 10th International Symposium on Artificial Intelligence and Mathematics*, 2007.

Peter Gregory, Maria Fox, and Derek Long.
A new empirical study of weak backdoors.
In *Principles and Practice of Constraint Programming*, pages 618–623. Springer, 2008.

Naomi Nishimura and Prabhakar Ragde.
Solving #SAT using vertex covers.
Acta Informatica, 44(7):509–523, 2007.

References III

Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider.
Detecting backdoor sets with respect to Horn and binary clauses.
Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004), 2004.

Igor Razgon and B. O'Sullivan.
Almost 2-SAT is fixed-parameter tractable.
Journal of Computer and System Sciences, 75(8):435–450, 2009.

References IV

Roberto Rossi, Steven Prestwich, S.A. Tarim, and Brahim Hnich.
Generalizing Backdoors.
In *Proceedings of the 5th International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS 2008)*, number 03, 2008.

Marko Samer.
Backdoor trees.
Proceedings of the 23rd Conference on Artificial, pages 363–368, 2008.

Stefan Szeider.
Backdoor sets for DLL subsolvers.
SAT 2005, pages 73–88, 2006.

References V

Ryan Williams, C.P. Gomes, and Bart Selman.
Backdoors to typical case complexity.
In *Proceeding of IJCAI-03*, volume 18, pages 1173–1178, 2003.