Backdoors to Satisfaction: Parameterized Complexity

Serge Gaspers

The University of New South Wales, Sydney, Australia
National ICT Australia, Sydney, Australia

First Symposium on Structure in Hard Combinatorial Problems
16-May-2013
Vienna, Austria
Outline

1. Backdoors
2. Parameterized Complexity
3. Detecting Backdoors
4. Tree-like SAT instances
5. Algorithm for detecting strong $\mathcal{W}_1$-backdoors
SAT and #SAT

**SAT**
- **Input:** A propositional formula $F$ in conjunctive normal form (CNF)
- **Question:** Is there an assignment to $\text{var}(F)$ satisfying all clauses of $F$?

**#SAT**
- **Input:** A CNF formula $F$
- **Question:** What is the number of assignments to $\text{var}(F)$ satisfying all clauses of $F$?

Example:

$$(x_1 \lor x_2) \land (\neg x_2 \lor x_3 \lor \neg x_4) \land (x_1 \lor x_4) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4)$$
SAT: theory vs. practice

**theory**

- NP-complete
- ETH: SAT cannot be solved in time $2^{o(n)}$
- Strong ETH: SAT cannot be solved in time $(2 - \epsilon)^n$ for any $\epsilon > 0$

**practice**

- Want to solve an NP-complete problem? Just encode into SAT and use a SAT solver
- Real-world instances with millions of variables and clauses
Backdoors

- Belief: real world instances have a “hidden structure” that makes them easy to solve
- Challenge: measure and identify this hidden structure
- One way: Backdoor = set of “key” variables that make it easy to solve the formula
Backdoors

- CNF formula $F$
- Set of variables $B \subseteq \text{var}(F)$
- For a truth assignment $\tau$ to $B$, the reduced formula $F[\tau]$ is obtained from $F$ by removing all clauses satisfied by $\tau$ and removing all remaining literals on $B$ from the other clauses.
- Base class $C$: class of poly-time solvable CNF formulas

**Definition (Weak Backdoor [Williams, Gomes, Selman, 2003])**

$B$ is a weak $C$-backdoor for $F$ if there is a truth assignment $\tau$ to $B$ such that $F[\tau] \in C$ and $F[\tau]$ is satisfiable.

**Definition (Strong Backdoor [Williams, Gomes, Selman, 2003])**

$B$ is a strong $C$-backdoor for $F$ if for every truth assignment $\tau$ to $B$ we have $F[\tau] \in C$. 
Backdoors

- CNF formula $F$
- Set of variables $B \subseteq \text{var}(F)$
- For a truth assignment $\tau$ to $B$, the \textit{reduced formula} $F[\tau]$ is obtained from $F$ by removing all clauses satisfied by $\tau$ and removing all remaining literals on $B$ from the other clauses
- Base class $C$: class of poly-time solvable CNF formulas

**Definition (Weak Backdoor [Williams, Gomes, Selman, 2003])**

$B$ is a \textit{weak $C$-backdoor} for $F$ if there is a truth assignment $\tau$ to $B$ such that $F[\tau] \in C$ and $F[\tau]$ is satisfiable.

**Definition (Strong Backdoor [Williams, Gomes, Selman, 2003])**

$B$ is a \textit{strong $C$-backdoor} for $F$ if for every truth assignment $\tau$ to $B$ we have $F[\tau] \in C$. 
Backdoors

- CNF formula $F$
- Set of variables $B \subseteq \text{var}(F)$
- For a truth assignment $\tau$ to $B$, the reduced formula $F[\tau]$ is obtained from $F$ by removing all clauses satisfied by $\tau$ and removing all remaining literals on $B$ from the other clauses
- Base class $C$: class of poly-time solvable CNF formulas

Definition (Weak Backdoor [Williams, Gomes, Selman, 2003])

$B$ is a **weak $C$-backdoor** for $F$ if there is a truth assignment $\tau$ to $B$ such that $F[\tau] \in C$ and $F[\tau]$ is satisfiable.

Definition (Strong Backdoor [Williams, Gomes, Selman, 2003])

$B$ is a **strong $C$-backdoor** for $F$ if for every truth assignment $\tau$ to $B$ we have $F[\tau] \in C$. 
Experimental results

Table 4. Size, percentage, and number of small backdoors found by the local search algorithms within a cutoff of 3 hours when applied to real-world instances with \( n \) variables (\( n < 10,000 \)) and \( m \) clauses.

<table>
<thead>
<tr>
<th>Instance</th>
<th>( n )</th>
<th>( m )</th>
<th>KILBY</th>
<th>KILBYIMP</th>
<th>TABU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>BD size (%)</td>
<td># BDs</td>
<td>BD size (%)</td>
</tr>
<tr>
<td>SAT Competition 2002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apex7_gr_rcs_w5.shuffled</td>
<td>1500</td>
<td>11136</td>
<td>77 (5.13%)</td>
<td>1</td>
<td>47 (3.13%)</td>
</tr>
<tr>
<td>dp10s10.shuffled</td>
<td>8372</td>
<td>8557</td>
<td>9 (0.11%)</td>
<td>10520</td>
<td>9 (0.11%)</td>
</tr>
<tr>
<td>bart11.shuffled</td>
<td>162</td>
<td>675</td>
<td>15 (9.26%)</td>
<td>4190</td>
<td>14 (8.64%)</td>
</tr>
<tr>
<td>SAT-Race 2005 and 2008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grieu-vmpc-s05-24s</td>
<td>576</td>
<td>49478</td>
<td>3 (0.52%)</td>
<td>143</td>
<td>3 (0.52%)</td>
</tr>
<tr>
<td>grieu-vmpc-s05-27r</td>
<td>729</td>
<td>71380</td>
<td>4 (0.55%)</td>
<td>710</td>
<td>4 (0.55%)</td>
</tr>
<tr>
<td>simon-mixed-s02bis-01</td>
<td>2424</td>
<td>13793</td>
<td>8 (0.33%)</td>
<td>566</td>
<td>8 (0.33%)</td>
</tr>
<tr>
<td>simon-s02b-r4b1k1.2</td>
<td>2424</td>
<td>13811</td>
<td>8 (0.33%)</td>
<td>394</td>
<td>7 (0.29%)</td>
</tr>
<tr>
<td>Blocks world planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw_large.c</td>
<td>3016</td>
<td>50237</td>
<td>4 (0.13%)</td>
<td>1934</td>
<td>3 (0.10%)</td>
</tr>
<tr>
<td>bw_large.d</td>
<td>6325</td>
<td>131607</td>
<td>6 (0.10%)</td>
<td>790</td>
<td>5 (0.08%)</td>
</tr>
<tr>
<td>Logistics planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>logistics.a</td>
<td>828</td>
<td>3116</td>
<td>20 (2.42%)</td>
<td>147</td>
<td>20 (2.42%)</td>
</tr>
<tr>
<td>logistics.b</td>
<td>843</td>
<td>3480</td>
<td>16 (1.90%)</td>
<td>1688</td>
<td>15 (1.78%)</td>
</tr>
<tr>
<td>logistics.c</td>
<td>1141</td>
<td>5867</td>
<td>26 (2.28%)</td>
<td>18</td>
<td>25 (2.19%)</td>
</tr>
<tr>
<td>logistics.d</td>
<td>4713</td>
<td>16588</td>
<td>25 (0.53%)</td>
<td>39</td>
<td>22 (0.47%)</td>
</tr>
</tbody>
</table>

[Li, van Beek, 2011] weak backdoors to UP+2CNF+1-VAL+0-VAL
Backdoor Problems

Weak (Strong) $C$-Backdoor Detection
Input: A CNF formula $F$, an integer $k$
Question: Does $F$ have a weak (strong) $C$-backdoor of size at most $k$?

Weak (Strong) $C$-Backdoor Evaluation
Input: A CNF formula $F$, a weak (strong) $C$-backdoor $B$
Question: Is $F$ satisfiable?
Outline

1. Backdoors
2. Parameterized Complexity
3. Detecting Backdoors
4. Tree-like SAT instances
5. Algorithm for detecting strong \( \mathcal{W}_1 \)-backdoors
"complexity is not governed by the instance size alone"

**Definition (Parameterized problem)**

A parameterized decision problem is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet $\Sigma$. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, $x$ is the main part and $k$ the parameter.

**FPT**: class of param. pbs that can be solved in time $f(k) \cdot n^{O(1)}$

**W[·]**: parameterized intractability classes

**XP**: class of param. pbs that can be solved in time $f(k) \cdot n^{g(k)}$

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq XP.$$  

All inclusions believed to be strict.
Parameterized Backdoor Problems

Weak (Strong) $C$-Backdoor Detection
Input: A CNF formula $F$, an integer $k$
Parameter: $k$
Question: Does $F$ have a weak (strong) $C$-backdoor of size at most $k$?

Weak (Strong) $C$-Backdoor Evaluation
Input: A CNF formula $F$, a weak (strong) $C$-backdoor $B$
Parameter: $k = |B|$
Question: Is $F$ satisfiable?
Weak Backdoor Detection

Simple Weak $C$-Backdoor Detection Algorithm

**Input:** A CNF formula $F$ and an integer $k$.

**Output:** YES if $F$ has a weak $C$-backdoor of size $k$, and NO otherwise.

```
foreach subset $B \subseteq \text{var}(F)$ with $|B| = k$ do
  foreach assignment $\tau: B \rightarrow \{0, 1\}$ do
    if $F[\tau] \in C$ then
      if $F[\tau]$ is satisfiable then
        return YES
  return NO
```

- run time: $\binom{n}{k} \cdot 2^k \cdot n^{O(1)} = n^{k+O(1)}$
- XP-algorithm
Weak Backdoor Detection

Simple Weak $C$-Backdoor Detection Algorithm

**Input:** A CNF formula $F$ and an integer $k$.

**Output:** YES if $F$ has a weak $C$-backdoor of size $k$, and NO otherwise.

```plaintext
foreach subset $B \subseteq \text{var}(F)$ with $|B| = k$ do
  foreach assignment $\tau : B \rightarrow \{0, 1\}$ do
    if $F[\tau] \in C$ then
      if $F[\tau]$ is satisfiable then
        return YES
  return NO
```

- run time: $\binom{n}{k} \cdot 2^k \cdot n^{O(1)} = n^{k+O(1)}$
- XP-algorithm
Simple Strong $C$-Backdoor Detection Algorithm

**Input:** A CNF formula $F$ and an integer $k$.

**Output:** YES if $F$ has a strong $C$-backdoor of size $k$, and NO otherwise.

```plaintext
foreach subset $B \subseteq \text{var}(F)$ with $|B| = k$ do
    valid ← true
    foreach assignment $\tau : B \rightarrow \{0, 1\}$ do
        if $F[\tau] \not\in C$ then
            valid ← false
    if valid then
        return YES
return NO
```

- run time: $\binom{n}{k} \cdot 2^k \cdot n^{O(1)} = n^{k+O(1)}$
- XP-algorithm
Strong Backdoor Detection

Simple Strong $C$-Backdoor Detection Algorithm

**Input:** A CNF formula $F$ and an integer $k$.

**Output:** Yes if $F$ has a strong $C$-backdoor of size $k$, and No otherwise.

```plaintext
foreach subset $B \subseteq \text{var}(F)$ with $|B| = k$ do
    valid ← true
    foreach assignment $\tau : B \rightarrow \{0, 1\}$ do
        if $F[\tau] \notin C$ then
            valid ← false
    if valid then
        return Yes
return No
```

- run time: $\binom{n}{k} \cdot 2^k \cdot n^{O(1)} = n^{k+O(1)}$
- XP-algorithm
Backdoor Evaluation

**Simple $C$-Backdoor Evaluation Algorithm**

**Input:** A CNF formula $F$ and a weak or strong $C$-backdoor $B$ of size $k$.

**Output:** Yes if $F$ is satisfiable, and No otherwise.

```
foreach assignment $\tau : B \to \{0, 1\}$ do
    if $F[\tau] \in C$ then /* not necessary for strong */
        if $F[\tau]$ is satisfiable then
            return Yes
    return No /* not possible for weak */
```

- run time: $2^k \cdot n^{O(1)}$
- FPT-algorithm
Backdoor Evaluation

**Simple C-Backdoor Evaluation Algorithm**

**Input:** A CNF formula $F$ and a weak or strong $C$-backdoor $B$ of size $k$.

**Output:** YES if $F$ is satisfiable, and NO otherwise.

```
foreach assignment $\tau : B \rightarrow \{0, 1\}$ do
    if $F[\tau] \in C$ then /* not necessary for strong */
        if $F[\tau]$ is satisfiable then
            return YES
    return NO /* not possible for weak */
```

- run time: $2^k \cdot n^{O(1)}$
- FPT-algorithm
Consequences for SAT

- The challenging part is Backdoor Detection.
- If Weak (Strong) $C$-Backdoor Detection is FPT, then SAT is FPT parameterized by the size of a smallest weak (strong) $C$-backdoor.
Outline

1. Backdoors
2. Parameterized Complexity
3. Detecting Backdoors
4. Tree-like SAT instances
5. Algorithm for detecting strong \( \mathcal{W}_1 \)-backdoors
Detecting backdoors to some base classes

<table>
<thead>
<tr>
<th>Base Class</th>
<th>Weak</th>
<th>Strong</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CNF</td>
<td>$r$-CNF</td>
</tr>
<tr>
<td>HORN</td>
<td>$W[2]$-h</td>
<td>FPT</td>
</tr>
<tr>
<td>2CNF</td>
<td>$W[2]$-h</td>
<td>FPT</td>
</tr>
</tbody>
</table>

The parameterized complexity of finding weak and strong backdoor sets of CNF formulas and $r$-CNF formulas, where $r \geq 3$ is a fixed integer.

What does this tell us?

**FPT cases**
- There is an algorithm with running time $f(k) \cdot n^{O(1)}$ that either finds a backdoor of size $k$, or determines that no such backdoor exists.
- If the instance has a small backdoor, there is at least one efficient way to find it (maybe many efficient ways).

**W[.]-hard cases**
- There is probably no algorithm with running time $f(k) \cdot n^{O(1)}$ that either finds a backdoor of size $k$, or determines that no such backdoor exists.
- There are instances with small backdoors of size $k$, but probably no efficient way to find these backdoors.
- Maybe a backdoor of size $k + 1$ can still be found efficiently... or one of size $2^k$?
FPT Approximation

**Definition ([Downey, Fellows, McCartin, 2006])**

A parameterized algorithm is an FPT-approximation algorithm for a minimization problem if there exist functions $f$, $g$ such that on input $(x, k)$, the algorithm has running time $f(k) \cdot n^{O(1)}$ and it either

- determines that $(x, k)$ is a NO-instance, or
- determines that $(x, k')$ is a YES-instance for some $k' \leq g(k)$
Incidence graph

Incidence graph of the formula $F = \bigwedge_{i=1}^{8} c_i$ with

$$c_1 = t \lor \neg u, \quad c_2 = u \lor v \lor w, \quad c_3 = w \lor x, \quad c_4 = x \lor \neg y,$$
$$c_5 = y \lor \neg z, \quad c_6 = t \lor u \lor \neg w, \quad c_7 = \neg x \lor z, \quad c_8 = \neg t \lor w \lor x$$
Acyclic SAT formulas

Definition

A SAT formula is **acyclic** if its incidence graph has no cycle.

Definition

**FOREST** denotes the class of all acyclic SAT formulas.
Results for FOREST-backdoors

Theorem ([Gaspers, Szeider, ICALP 2012])


Theorem ([Gaspers, Szeider, ICALP 2012])

For every constant $r \geq 3$, Weak FOREST-Backdoor Detection is FPT for $r$-CNF formulas.

Theorem ([Gaspers, Szeider, ICALP 2012])

There is an FPT-approximation algorithm for Strong FOREST-Backdoor Detection.
Consequences for SAT

Corollary ([Gaspers, Szeider, ICALP 2012])

\[ r\text{-SAT and } r\text{-}\#\text{SAT are FPT parameterized by the size of a smallest weak FOREST-backdoor.} \]

Corollary ([Gaspers, Szeider, ICALP 2012])

\[ \text{SAT and } \#\text{SAT are FPT parameterized by the size of a smallest strong FOREST-backdoor.} \]
More general?

Are there larger base classes with an FPT-approximation for Strong Backdoor Detection?
Tree decompositions (by example)

- A graph $G$

  $\begin{align*}
  a & \xrightarrow{b} c \\
  & \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\
  & \quad c \quad d \quad h \quad i \quad j \\
  e & \quad f \quad g \\
  & \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\
  & \quad e \quad f \quad h \quad i \quad j \quad k
  \end{align*}$

- A tree decomposition of $G$

  $\begin{align*}
  a, b, c & \quad c, d, e \quad d, e, f \quad d, f, h \quad h, i \\
  & \quad \quad \quad f, g \quad i, j \quad i, k
  \end{align*}$
Tree decompositions (by example)

- A graph $G$

- A tree decomposition of $G$
Tree decompositions (by example)

- A graph $G$

\[ a \rightarrow b \leftarrow c \rightarrow d \leftarrow h \rightarrow i \rightarrow j \]
\[ e \rightarrow f \leftarrow g \]

- A tree decomposition of $G$

\[ a, b, c \rightarrow c, d, e \rightarrow d, e, f \rightarrow d, f, h \rightarrow h, i \rightarrow i, j \rightarrow i, k \]

Conditions:
Tree decompositions (by example)

- A graph $G$

- A tree decomposition of $G$

Conditions: covering
Tree decompositions (by example)

- A graph $G$

\[ a \rightarrow b \rightarrow c \rightarrow d \rightarrow h \rightarrow i \rightarrow j \]

\[ e \rightarrow f \rightarrow g \]

- A tree decomposition of $G$

\[ a, b, c \rightarrow c, d, e \rightarrow d, e, f \rightarrow d, f, h \rightarrow h, i \rightarrow i, j \rightarrow i, k \]

Conditions: covering and connectedness.
Let $G$ be a graph, $T$ a tree, and $\chi$ a labeling of the nodes of $T$ by subsets of $V(G)$.

We refer to the sets $\chi(t)$ as “bags”.

The pair $(T, \chi)$ is a tree decomposition of $G$ if the following two conditions hold:

- For every edge $vw \in E(G)$ there exists a node $t$ of $T$ such that $v, w \in \chi(t)$ (“covering”).
- For every vertex $v$ of $G$, the graph $T[t \in V(T) : v \in \chi(t)]$ is a non-empty (connected) tree (“connectedness”).
**Treewidth**

- The **width** of a tree decomposition \((T, \chi)\) is defined as the maximum \(|\chi(t)| - 1\) over all nodes \(t\) of \(T\).
- The **treewidth** \(tw(G)\) of a graph \(G\) is the minimum width over all its tree decompositions.
Trees have treewidth 1.
Cycles have treewidth 2.
The complete graph on $n$ vertices has treewidth $n - 1$. 
Treewidth of SAT formulas

- A CNF formula has treewidth $t$ if its incidence graph has treewidth $t$.
- $\mathcal{W}_t$ denotes the class of all CNF formulas with treewidth at most $t$. 
Outline

1. Backdoors
2. Parameterized Complexity
3. Detecting Backdoors
4. Tree-like SAT instances
5. Algorithm for detecting strong \( \mathcal{W}_t \)-backdoors
Outline of the FPT approximation algorithm

**Algorithm TW-backdoor**

**Input:** A CNF formula $F$ and integers $k, t \geq 0$.

**Output:** A strong $\mathcal{W}_t$-backdoor of $F$ of size $\leq 2^k$, or NO if $F$ has no strong $\mathcal{W}_t$-backdoor of size $k$.

- **if** $F$ has “small” treewidth [Bodlaender, 1996] **then**
  - Express the problem in MSO$_2$ using [Adler, Grohe, Kreutzer, 2008] [Lagergren, 1998]
  - Use Courcelle’s theorem [Courcelle, 1990] [Arnborg, Lagergren, Seese, 1991]

- **else**
  - …
Outline of the FPT approximation algorithm

Algorithm TW-backdoor

**Input:** A CNF formula $F$ and integers $k, t \geq 0$.

**Output:** A strong $\mathcal{W}_t$-backdoor of $F$ of size $\leq 2^k$, or No if $F$ has no strong $\mathcal{W}_t$-backdoor of size $k$.

```plaintext
... else
  Compute a large wall as a topological minor [Robertson, Seymour, Thomas ’94] [Grohe, Kawarabayashi, Marx, Wollan ’11]
  Compute a set $S$ of $f(k, t)$ variables such that every strong $\mathcal{W}_t$-backdoor contains at least one of these variables
  foreach $x \in S$ do
    $B_x \leftarrow$ TW-backdoor($F[x = 1], k - 1, t$)
    $B_{\neg x} \leftarrow$ TW-backdoor($F[x = 0], k - 1, t$)
    if ($B_x \neq \text{No}$) \& (B_{\neg x} \neq \text{No}) then
      return $B_x \cup B_{\neg x} \cup \{x\}$
  return No
```
(Topological) Minors

**Definition ((Topological) Minor)**

Let $H, G$ be two graphs. $H$ is a (topological) minor of $G$ if a graph isomorphic to $H$ can be obtained from $G$ by a sequence of the following operations:

- delete a vertex
- delete an edge
- contract an edge (incident to a vertex of degree 2)

![Diagram](Contract the edge $uv$)
Obstructions for $\mathcal{W}_3$

- $W_8$
- $K_5$
- $K_{4,4}$
- $(1,8)$
- $(8,8)$
- $(1,1)$
- $(8,1)$
Using the Topological Wall Minor

- Large wall as a topological minor $\rightarrow$ many disjoint wall obstructions
- Each obstruction needs to be killed
Internal and External Killers

- An internal killer
- An external killer

At most $k$ wall obstructions are killed internally.

⇒ “Guess” them and discard them

All remaining obstructions are killed externally
Recall: we have many disjoint wall obstructions, and all of them need to be killed externally.

- $\geq 1/2^k$-th of all wall obstructions are killed externally by the same backdoor variables
- $\Rightarrow$ “Guess” this subset $O$ of wall obstructions and the number $\ell$ of backdoor variables that kill them externally
- Denote by $Z$ the set of common external killers of the wall obstructions in $O$

Aim: Find a small subset $S \subseteq Z$ such that every valid (i.e., respecting our guesses) strong $\mathcal{W}_t$-backdoor contains a vertex from $S$. Then, $S$ can be used for branching.
Recall: we have many disjoint wall obstructions, and all of them need to be killed externally.

- $\geq 1/2^k$-th of all wall obstructions are killed externally by the same backdoor variables
- $\Rightarrow$ “Guess” this subset $\mathcal{O}$ of wall obstructions and the number $\ell$ of backdoor variables that kill them externally
- Denote by $Z$ the set of common external killers of the wall obstructions in $\mathcal{O}$

Aim: Find a small subset $S \subseteq Z$ such that every valid (i.e., respecting our guesses) strong $\mathcal{W}_t$-backdoor contains a vertex from $S$. Then, $S$ can be used for branching.
We have 3 rules to construct $S$:

**Rule 1 (Few Common Killers).** If $|Z| \leq 6knb(t)$, then set $S := Z$. 
($nb(t) = \lceil 16(t + 2)\log(t + 2) \rceil$)

**Rule 2 (Multiple Neighborhoods).** If there is a subset $L \subseteq Z$ such that $L$ is the neighborhood of at least $t2^\ell + 1$ vertices in $B_m(O)$, then set $S := L$.

**Rule 3 (No Multiple Neighborhoods).** Set $S$ to be the $6knb(t)$ vertices from $Z$ of highest degree in $B(O)$ (ties are broken arbitrarily).

But what are $B_m(O)$ and $B(O)$?
We have 3 rules to construct $S$:

**Rule 1 (Few Common Killers).** If $|Z| \leq 6 knb(t)$, then set $S := Z$. \[(nb(t) = \lceil 16(t + 2)\log(t + 2) \rceil)\]

**Rule 2 (Multiple Neighborhoods).** If there is a subset $L \subseteq Z$ such that $L$ is the neighborhood of at least $t2^\ell + 1$ vertices in $B_m(\mathcal{O})$, then set $S := L$.

**Rule 3 (No Multiple Neighborhoods).** Set $S$ to be the $6knb(t)$ vertices from $Z$ of highest degree in $B(\mathcal{O})$ (ties are broken arbitrarily).

But what are $B_m(\mathcal{O})$ and $B(\mathcal{O})$?
We have 3 rules to construct $S$:

**Rule 1 (Few Common Killers).** If $|Z| \leq 6knb(t)$, then set $S := Z$.

$$nb(t) = \lceil 16(t + 2)\log(t + 2) \rceil$$

**Rule 2 (Multiple Neighborhoods).** If there is a subset $L \subseteq Z$ such that $L$ is the neighborhood of at least $t2^\ell + 1$ vertices in $B_m(O)$, then set $S := L$.

**Rule 3 (No Multiple Neighborhoods).** Set $S$ to be the $6knb(t)$ vertices from $Z$ of highest degree in $B(O)$ (ties are broken arbitrarily).

But what are $B_m(O)$ and $B(O)$?
We have 3 rules to construct $S$:

**Rule 1 (Few Common Killers).** If $|Z| \leq 6\text{nb}(t)$, then set $S := Z$. 
($\text{nb}(t) = \lceil 16(t + 2)\log(t + 2) \rceil$)

**Rule 2 (Multiple Neighborhoods).** If there is a subset $L \subseteq Z$ such that $L$ is the neighborhood of at least $t2^\ell + 1$ vertices in $B_m(O)$, then set $S := L$.

**Rule 3 (No Multiple Neighborhoods).** Set $S$ to be the $6\text{nb}(t)$ vertices from $Z$ of highest degree in $B(O)$ (ties are broken arbitrarily).

But what are $B_m(O)$ and $B(O)$?
We have 3 rules to construct $S$:

**Rule 1 (Few Common Killers).** If $|Z| \leq 6knb(t)$, then set $S := Z$. 
($nb(t) = \lceil 16(t + 2)\log(t + 2) \rceil$)

**Rule 2 (Multiple Neighborhoods).** If there is a subset $L \subseteq Z$ such that $L$ is the neighborhood of at least $t2^\ell + 1$ vertices in $B_m(O)$, then set $S := L$.

**Rule 3 (No Multiple Neighborhoods).** Set $S$ to be the $6knb(t)$ vertices from $Z$ of highest degree in $B(O)$ (ties are broken arbitrarily).

But what are $B_m(O)$ and $B(O)$?
Definition (obstruction-template)

An obstruction-template $OT(W)$ of a wall-obstruction $W \in O$ is a triple $(B(W), P, R)$, where

- $B(W)$ is a bipartite graph whose vertex set is bipartitioned into the two independent sets $Z$ and $Q_W$, where $Q_W$ is a set of new vertices,
- $P$ is a partition of $V(W)$ into regions such that for each region $A \in P$, we have that $W[A]$ is connected, and
- $R : Q_W \to P$ is a function associating a region of $P$ with each vertex in $Q_W$. 
The Beast (2)

Definition (valid obstruction-template)

An obstruction-template $OT(W) = (B(W), P, R)$ of a wall-obstruction $W \in O_s$ is valid if it satisfies the following properties:

1. **only existing edges:** for each $q \in Q_W$ we have that $N_{B(W)}(q) \subseteq N_G(R(q))$,

2. **private neighbor:** for each $q \in Q_W$, there is a vertex $z \in N_{B(W)}(q)$, called $q$’s **private neighbor**, such that there is no other $q' \in N_{B(W)}(z)$ with $R(q') = R(q)$,

3. **degree-$Z$:** for each $z \in Z$ we have that $d_{B(W)}(z) \geq 1$,

4. **degree-$Q_W$:** for each $q \in Q_W$ we have that $\text{nb}(t) \leq d_{B(W)}(q) \leq 3\text{nb}(t)$, and

5. **vulnerable vertex:** for each $q \in Q_W$, there is at most one vertex $v \in R(q)$, called $q$’s **vulnerable vertex**, such that $N_G(v) \cap Z \not\subseteq N_{B(W)}(q)$. 
\( \mathcal{B}_m(\mathcal{O}) \) is obtained by taking the union of all \( \mathcal{B}(W), W \in \mathcal{O} \).

\( \mathcal{B}(\mathcal{O}) \) is obtained from \( \mathcal{B}_m(\mathcal{O}) \) by merging vertices from \( V(\mathcal{B}_m(\mathcal{O})) \setminus Z \) with identical neighborhoods.
Computing and Taming the Beast

To identify a small $S \subseteq Z$ intersecting every valid strong $\mathcal{W}_t$-backdoor, we need to find obstructions involving $S$ and $O$ for at least one assignment to every candidate backdoor of size $k$ avoiding $S$.

Valid obstruction-templates model various ways to assemble such obstructions.

A valid obstruction-template can be computed in $O(n^2)$ time.

We prove that for a set $S$ constructed by our rules, a valid $\mathcal{W}_t$-backdoor contains a variable from $S$, otherwise at least one assignment to the backdoor produces a formula whose incidence graph has treewidth at least $t + 1$. 
To identify a small $S \subseteq Z$ intersecting every valid strong $\mathcal{W}_t$-backdoor, we need to find obstructions involving $S$ and $O$ for at least one assignment to every candidate backdoor of size $k$ avoiding $S$.

Valid obstruction-templates model various ways to assemble such obstructions.

A valid obstruction-template can be computed in $O(n^2)$ time.

We prove that for a set $S$ constructed by our rules, a valid $\mathcal{W}_t$-backdoor contains a variable from $S$, otherwise at least one assignment to the backdoor produces a formula whose incidence graph has treewidth at least $t + 1$. 

To identify a small $S \subseteq Z$ intersecting every valid strong $\mathcal{W}_t$-backdoor, we need to find obstructions involving $S$ and $\mathcal{O}$ for at least one assignment to every candidate backdoor of size $k$ avoiding $S$.

Valid obstruction-templates model various ways to assemble such obstructions.

A valid obstruction-template can be computed in $O(n^2)$ time.

We prove that for a set $S$ constructed by our rules, a valid $\mathcal{W}_t$-backdoor contains a variable from $S$, otherwise at least one assignment to the backdoor produces a formula whose incidence graph has treewidth at least $t + 1$. 
To identify a small $S \subseteq Z$ intersecting every valid strong $\mathcal{W}_t$-backdoor, we need to find obstructions involving $S$ and $O$ for at least one assignment to every candidate backdoor of size $k$ avoiding $S$.

Valid obstruction-templates model various ways to assemble such obstructions.

A valid obstruction-template can be computed in $O(n^2)$ time.

We prove that for a set $S$ constructed by our rules, a valid $\mathcal{W}_t$-backdoor contains a variable from $S$, otherwise at least one assignment to the backdoor produces a formula whose incidence graph has treewidth at least $t + 1$. 

Computing and Taming the Beast
Results for Bounded Treewidth

**Theorem ([Gaspers, Szeider, 2012])**

There is an FPT algorithm with parameter $k + t$ that either concludes that $F$ has no strong $\mathcal{W}_t$-backdoor of size at most $k$ or finds a strong $\mathcal{W}_t$-backdoor of $F$ of size at most $2^k$.

**Corollary ([Gaspers, Szeider, 2012])**

There is a cubic-time algorithm that, given a CNF formula $F$, computes the number of satisfying assignments of $F$ or concludes that the smallest strong $\mathcal{W}_t$-backdoor of $F$ is larger than $k$, for any pair of constants $k, t \geq 0$. 
Related Results

- Faster and simpler randomized FPT algorithm for detecting weak FOREST-backdoors for $r$-CNF formulas
  (based on [Fomin, Lokshtanov, Misra, Saurabh, FOCS 2012])
- Also extends to the base class $\mathcal{W}_f \cap r$-CNF
Conclusion

- Aim at explaining the good running times of SAT solvers
- Is there a strong correlation between 
  “the problem is FPT w.r.t. parameter $k$”
  and 
  “heuristics work well if $k$ is small”? 
- Need simpler algorithms (randomization?) 
- Is Strong FOREST/$\mathcal{W}_i$-backdoor detection FPT? 
- Combination of base classes
Thank you!

Questions? Comments?