Measuring the Hardness of SAT Instances*

Carlos Ansótegui
DIEI, UdL
Jaume II, 69, 25001 Lleida, Spain
carlos@diei.udl.cat

María Luisa Bonet
LSI, UPC
J. Girona, 1-3, 08034 Barcelona, Spain
bonet@lsi.upc.edu

Jordi Levy and **Felip Manyà**
IIIA, CSIC
Campus UAB, 08193 Bellaterra, Spain
{levy,felip}@iiia.csic.es

Abstract

The search of a precise measure of what hardness of SAT instances means for state-of-the-art solvers is a relevant research question. Among others, the space complexity of tree-like resolution (also called hardness), the minimal size of strong backdoors and of cycle-cutsets, and the treewidth can be used for this purpose.

We propose the use of the tree-like space complexity as a solid candidate to be the best measure for solvers based on DPLL. To support this thesis we provide a comparison with the other mentioned measures. We also conduct an experimental investigation to show how the proposed measure characterizes the hardness of random and industrial instances.

Introduction

Even though the SAT/CSP problems are NP-Complete, there are groups of instances that can be solved quickly by state-of-the-art solvers. For example, industrial instances may have a huge a number of variables and still be solved in a reasonable amount of time by modern solvers. Therefore, a better knowledge about what a hard/easy instance is, would help to design better practical SAT solvers. This question is related with the characterization of *real-world* (industrial) instances. This was one of the motivations of (Williams, Gomes, and Selman 2003; Dilkina, Gomes, and Sabharwal 2007), when they defined backdoors as a measure of hardness. They answer two key questions: (1) What is the size of the backdoors in real-world (industrial) instances? Experimentally they conclude that they are small. (2) Even taking into account the expense of searching for backdoors, can one still obtain an overall computational advantage using them? They prove that, for constant-bounded backdoor size, there exists a polynomial decision algorithm, being the size of the backdoor its degree. In CSP there are two notions that can also characterize the hardness of problems: the size of cycle-cutsets, and the treewidth. These two notions also share the good properties of backdoors: when they are constant-bounded, they measure the degree of a polynomial decision algorithm.

In this paper we propose the use of another measure: the *space complexity of tree-like resolution* (Torán 1999; Esteban and Torán 1999; 2001; Ben-Sasson and Galesi 2003) also called *hardness of a formula* in (Kullmann 1999; 2004). In this paper we will call it *space* for short. Like for the previous measures, we also prove that constant-bounded space implies the existence of a polynomial decision algorithm, being the space the degree of the polynomial (see Theorem 6). Moreover, we prove that the space is smaller than the size of cycle-cutsets and, under certain assumptions, also smaller than the size of backdoors.

Our project can be formalized as:

Find a measure ψ, and an algorithm that given a formula Γ decides satisfiability in time $\mathcal{O}(n^{\psi(\Gamma)})$. The smaller the measure is, the better it characterizes the hardness of a formula.

The preference for smaller measures is because a bigger measure means that some instances, that could be solved efficiently, are erroneously classified as hard. Therefore, according to this project, the space characterizes the hardness of problems better than strong backdoors and cycle-cutsets. We also show that there are cases where the space is arbitrarily smaller than the treewidth (one is 2 whereas the other is equal to the number of variables).

We know that the *width of a formula* is smaller than the space (Esteban and Torán 2001; Nordström 2006), and that there exists an algorithm that works in time $\mathcal{O}(n^{width(\Gamma)})$. Therefore, we could conclude that, according to our project, the width is a better measure than the space. Unfortunately, this algorithm also requires space $\mathcal{O}(n^{width(\Gamma)})$ which disables its use in real solvers, and makes the width a bad measure of what is hard for those real solvers.

Another natural candidate measure is the logarithm of the minimum tree-like resolution proof of a formula, noted $ts(\Gamma)$. If a SAT solver tries to construct tree-like resolution proofs, the size of the minimal of such proofs will be a good measure of the hardness of the formula. Moreover, there is an algorithm that works in time $\mathcal{O}(n^{\log ts(\Gamma)})$ (Beame and Pitassi 1996). However, we also prove that the space is smaller than $\log ts(\Gamma)$. The more unbalanced the proof tree is, the bigger the difference between the two measures is.

*Research partially founded by the research projects TIN2006-15662-C02-02, and TIN2007-68005-C04-01/02/03, CONSOLIDER CSD2007-0022, INGENIO 2010, and the grant 2005-SGR-00093. The first author was partially supported by the program José Castillejo.

As a first step in the formulated research project we can justify the optimality of our measure respect to solvers based on tree-like resolution[1]. We use the following argumentation: a problem is *easy* for a solver based on tree-like resolution, if there is a *small measure* such that, instantiating one of the variables (smartly selected) by one truth value, the measure strictly decreases, and instantiating by the other truth value, it doesn't increase. This is the minimum requirement for ensuring polynomial solvability (in the measure) by tree search (being the degree of the polynomial the value of the measure). This is precisely the definition of space. Contrarily, in the definition of strong backdoors we force the measure to decrease in *both* sub-trees, and the selected variables to be the same in all the branches (the ones belonging to the backdoor). Instead, for space, selected variables don't have to be the same in all branches.

The fact we talk about tree-like proofs in our definition of space is not a big issue, despite the fact that most modern SAT solvers are not simply DPLL. This is because our notion of space is compared with other measures in terms of a project goal where nothing is said about the proof system.

The Space of an Unsatisfiable Formula

The tree-like space complexity of a formula is defined in (Esteban and Torán 1999; 2001) as the space needed to construct a tree-like resolution proof of the formula. Being the space the maximum number of clauses that have to be kept in memory simultaneously. The definition is not very intuitive, thus we give here an equivalent definition based on the minimum Horton-Strahler number of a tree-like resolution proof of the formula.

The *Horton-Strahler number of a tree* (Strahler of a tree, for short) (Horton 1945; Strahler 1952) was originally defined in the area of geology to study the morphology of rivers. Later, it was re-invented, in the area of computer science (Ershov 1958), as the minimum number of registers needed by a CPU to evaluate an arithmetic expression (built up from binary operations) and represented as a binary tree[2]. The notion is quite natural, but sometimes badly known, which has propitiated that it was re-invented several times. It is called d_c in (Esteban and Torán 1999) and *leveled height* in (Kullmann 1999). These are three equivalent definitions of the Strahler of a tree.

Definition 1 *(1) The Strahler of a (binary) tree is defined recursively as follows.*

$$hs(\bullet) = 0$$

$$hs\left(T_1 \overset{\bullet}{\swarrow\!\searrow} T_2\right) = \begin{cases} hs(T_1)+1 & \text{if } hs(T_1) = hs(T_2) \\ \max\{hs(T_1), hs(T_2)\} & \text{otherwise} \end{cases}$$

where $\bullet$ is a node and T_1 and T_2 are trees.
(2) The Strahler of a tree T is the depth of the biggest complete tree (i.e. tree where all branches have the same length) that can be embedded into T.

[1] A second step would be to define a measure and the corresponding algorithms for solvers based on learning.

[2] In fact the minimum number of register is the Strahler plus one.

(3) The Strahler of a tree is the minimum number of pointers (memory) that we need in order to traverse it, minus one.

The Strahler of a tree is bounded by its maximal depth, and by the logarithm of its size, and the three measures are equal in perfectly balanced trees.

Definition 2 *The space of a unsatisfiable CNF formula Γ, noted $s(\Gamma)$, is equal to the minimum among all the Strahlers of tree-like refutations of the formula.*

Notice that the space of a formula is bounded by the number of variables. However, as observed in the experiments, in real-world instances, the space is quite small, compared with the number of variables.

Lemma 3 *The space satisfies the following three properties*
(1) $s(\Gamma \cup \{\square\}) = 0$
(2) For any unsatisfiable formula Γ, and any partial truth assignment ϕ, we have $s(\phi(\Gamma)) \leq s(\Gamma)$.
(3) For any unsatisfiable formula Γ, if $\square \notin \Gamma$, then there exists a variable x and an assignment $\phi : \{x\} \to \{0,1\}$, such that $s(\phi(\Gamma)) \leq s(\Gamma) - 1$.

The space of a formula is the minimum measure on formulas that satisfy (1), (2) and (3). In other words, we could define the space as:[3]

$$s(\Gamma) = \min_{\substack{x,\overline{x}\in\Gamma \\ b\in\{0,1\}}} \{\, \max\{s([x\mapsto b](\Gamma))+1\,,\ s([x\mapsto \overline{b}](\Gamma))\}\,\}$$

when $\square \notin \Gamma$, and $s(\Gamma \cup \{\square\}) = 0$.

PROOF: (1), (2) and (3) are easy to prove. For the last part, suppose that x and b are the values that give us the minimal. W.l.o.g. assume that $b = 1$. From the proof tree of $[x \mapsto b](\Gamma) \vdash \square$, adding the literal $\overline{x}$ in the clauses where $[x \mapsto b]$ has removed it, but preserving the structure of the proof tree, we get a proof of $\Gamma \vdash \overline{x}$. Since we preserve the structure, we also preserve the Strahler of the tree. We proceed similarly for $[x \mapsto \overline{b}](\Gamma) \vdash \square$. Adding a cut of x to these two tree proofs, we get a proof of $\Gamma \vdash \square$, where the Strahler is the maximum between one of the original Strahlers, and the other plus one. Hence, it satisfies the equality. ■

Lemma 3 helps us understand the good properties of the notion of space: when we are forced to try two distinct assignments for a variable, if we are smart enough selecting the variable, we can get, in at least one of the sub-trees, a formula with a strictly smaller space (and the same or smaller in the other). This ensures that, if the space is small, we will avoid the combinatorial explosion of the worst-case.

Instances with Small and Big Space

The DPLL algorithm, as well as all its sequels, introduce important features that make the solvers more efficient than a simple blind check of all possible variable instantiations. In particular, they use the so-called "unit clause propagation". The following lemma characterizes the set of clauses that

[3] Notice that, since Γ is unsatisfiable, it either contains $\square$ or it contains a variable with both signs.

can be proved unsatisfiable only using unit propagation, as the formulas with space one. This set of clauses is equal to the set of *Horn renamable* clauses (Henschen and Wos 1974), i.e. the set of clauses that using a renaming of variables can be transformed into Horn clauses.

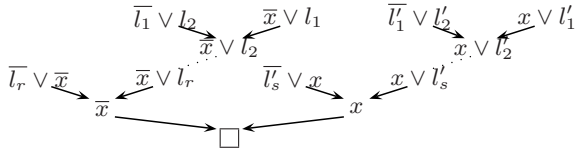
Lemma 4 *A formula* Γ *is Horn renamable iff* $s(\Gamma) \leq 1$.

PROOF: The equivalence between Horn renamable clauses and formulas that can be proved only using unit propagation is found in (Henschen and Wos 1974). And the equivalence between these ones and formulas that can be proved by linear tree-like resolution is found in (Beame, Kautz, and Sabharwal 2004). Now, it is easy to see that linear tree-like resolution proofs have space equal to one. ■

Lemma 5 *For every unsatisfiable* 2*-CNF formula* Γ *we have* $s(\Gamma) \leq 2$.

PROOF: If a 2-CNF formula is unsatisfiable, then either it can be proved with unit clause propagation, or we have a cycle of implications:
$x \to l_1 \to \cdots \to l_r \to \overline{x} \to l'_1 \to \ldots \to l'_s \to x$
for some variable x and a list of pair-wise distinct literals (variables with sign) $l_1, \ldots, l_r$ and a similar list $l'_1, \ldots, l'_s$.

In such case, we can construct the following proof with space two:



■

Ben-Sasson and Galesi (2003) prove that for random k-CNF formulas over n variables, with $\alpha\, n$ clauses (being $\alpha < 4.506$), with probability tending to 1 as n tends to ∞, resolution requires space $\Omega(\sqrt{\frac{n}{\alpha}} \frac{\log \frac{n}{\alpha}}{2 \log n})$. Since the space for general resolution is smaller than or equal to the space for tree-like resolution, this big lower bound explains why random formulas are difficult SAT instances. Likewise, (Torán 1999; Esteban and Torán 2001) show that the pigeonhole principle formulas PHP_n^{n+1} require space $\Omega(n)$.

Proving Instances with Small Space

The algorithm that we propose is a variation of the Beame-Pitassi algorithm (Beame and Pitassi 1996), that can be also found in (Kullmann 1999). Given an unsatisfiable formula, the algorithm searches its space, starting with space equals to one. This is done in the function beame_pitassi. For each particular possible space it applies the procedure try_strahler. In this procedure we cycle trough all $2\,n$ possible literals trying to recursively figure out if the formula resulting from falsifying this literal has space one less. Once we find such a literal, we recursively call try_strahler only once with the opposite literal and the same space. If a satisfying assignment is found with some of the instantiations, the algorithm aborts execution.

Theorem 6 *Satisfiability of a CNF formula* Γ *can be decided in time* $\mathcal{O}(n^{s(\Gamma)+1})$.

```
function try_strahler(Γ, s, φ) returns ⟨bool, prooftree⟩
  if s = 0 then return ⟨false, _⟩
  if φ falsifies a clause C ∈ Γ
  then return ⟨true, hypothesis(C)⟩
  if φ satisfies all clauses of Γ
  then print φ
       return ⟨true, _⟩
  foreach variable x ∉ domain(φ) and b ∈ {true, false} do
    ⟨found, t1⟩ = try_strahler(Γ, s − 1, φ ∪ [x ↦ b])
    if found
    then ⟨found, t2⟩ = try_strahler(Γ, s, φ ∪ [x ↦ ¬b])
         return ⟨found, cut(x, t1, t2)⟩
  endfor
  return ⟨false, _⟩
endfunction

function beame_pitassi(Γ)
  proved := false
  s := 1
  while s ≤ numvarsof(Γ) and ¬proved do
    ⟨proved, proof⟩ := try_strahler(Γ, s, [ ])
    s := s + 1
  endwhile
  exit ⟨unsat, proof⟩
endfunction
```

Figure 1: The decision algorithm.

PROOF: Using the algorithm of Fig. 1. Let $T(s,n)$ be the worst-case time needed by the try_strahler function to check a formula with n variables and space s. We can establish the recurrence $T(s,n) \leq 2\,n\,T(s-1,n-1) + T(s,n-1)$. The solution of this recurrence can be found in (Beame and Pitassi 1996), and is $T(s,n) = O(n^s)$. For the beame_pitassi function the worst-case time needed is $\sum_{i=1}^{s} \mathcal{O}(n^i) = O(n^{s+1})$. ■

In DPLL we proceed by selecting a variable, instantiating it by a truth value, and later by the contrary, generating two sub-trees in our proof search tree. This process, in the worst case, generates an exponential refutation proof tree, on the number of variables. In the absence of backjumping, clause learning, and other features of modern solvers, the *proof-search tree* has a size similar to the size of the *proof tree* resulting form the search. In some cases, it would be worth searching for a smaller proof tree, even on the expense of having a proof search tree bigger than the proof tree. In fact, this is the case with the Beame-Pitassi algorithm:

The size of the proof tree computed by the Beame-Pitassi algorithm is dominated by the recursion:

$$S(s,n) \leq S(s-1,n-1) + S(s,n-1) + 1$$

with solution $S(s,n) = 2\sum_{i=1}^{s} \binom{n}{i} = \mathcal{O}(n^s)$. For small values of s, this upper bound is tight. Therefore, for formulas with small space, the size of the proof tree $\mathcal{O}(n^s)$, and the time needed by the algorithm to find it $O(n^{s+1})$ are similar. However, for s close to n, the size of the proof tree is $\mathcal{O}(2^n)$, and the time needed by the algorithm to find it $\mathcal{O}(n^n)$. This last upper bound is bigger than the $\mathcal{O}(2^n)$ needed by DPLL, which is more competitive than Beame-Pitassi for instances with big space.

Like the question of the computation of backdoors (Dilkina, Gomes, and Sabharwal 2007), an interesting question here is the study of heuristics leading to the construction of the proof tree with smallest Strahler.

The Space Needed for Satisfiable Formulas

There are several natural extensions of the space definition to satisfiable formulas. Here we propose two of them, respectively denoted by s^a and s^b. The following are some standard and preliminary definitions.

Definition 7 *Let ϕ be a partial assignment, we define the clause associated to ϕ, noted C_ϕ, as $\bigvee_{\phi(x)=0} x \vee \bigvee_{\phi(y)=1} \overline{y}$. Given a formula Γ and a partial assignment ϕ, the formula $\phi(\Gamma)$ is the set resulting from Γ after removing all clauses containing a literal x such that $\phi(x) = 1$, and removing all literals x such that $\phi(x) = 0$.*
A partial assignment ϕ is said to satisfy a formula Γ if $\phi(\Gamma) = \emptyset$.

Definition 8 *Let Γ be a satisfiable or unsatisfiable formula, we propose the following two alternative definitions of generalized space:*

$$s^a(\Gamma) = \max_{\textit{assignment } \phi}\{s(\phi(\Gamma)) \mid \phi(\Gamma) \vdash \square\}$$
$$s^b(\Gamma) = s(\Gamma \cup \{C_\phi \mid \phi(\Gamma) = \emptyset\})$$

Notice that both definitions are extensions of the space definition for unsatisfiable formulas: if Γ is unsatisfiable, then $s(\Gamma) = s^a(\Gamma) = s^b(\Gamma)$. We can also prove the following results.

Lemma 9 *For any formula Γ, we have $s^a(\Gamma) \leq s^b(\Gamma)$*

PROOF: Sketch: Let $\hat{\phi}$ be an assignment maximizing $\{s(\phi(\Gamma)) \mid \phi(\Gamma) \vdash \square\}$. Hence, $s^a(\Gamma) = s(\hat{\phi}(\Gamma))$ and $\hat{\phi}(\Gamma) \vdash \square$. Now, since $\hat{\phi}$ transform the clauses C_ϕ of the definition of s^b into tautologies, we have $s(\hat{\phi}(\Gamma)) = s(\hat{\phi}(\Gamma \cup \{C_\phi \mid \phi(\Gamma) = \emptyset\}))$. Finally, from

$$s(\hat{\phi}(\Gamma \cup \{C_\phi \mid \phi(\Gamma) = \emptyset\})) \leq s(\Gamma \cup \{C_\phi \mid \phi(\Gamma) = \emptyset\}) = \overset{b}{s}(\Gamma)$$

we deduce the inequality. ■

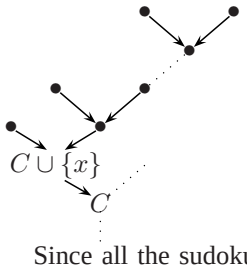
Lemma 10 *The adaptation of the Beame-Pitassi algorithm of Figure 1, given a satisfiable formula Γ, finds in time $\mathcal{O}(n^{s^b(\Gamma)})$ a complete set of satisfying assignments.*

PROOF: Sketch: The algorithm does a similar work for Γ and for $\Gamma \cup \{C_\phi \mid \phi(\Gamma) = \emptyset\}$. The only difference is that, in one case uses the clauses from $\{C_\phi \mid \phi(\Gamma) = \emptyset\}$ to get a contradiction, whereas in the other case prints the satisfying assignment. ■

An example of *easy* satisfiable formulas are commercial sudokus encoded as SAT formulas. The complexity of this problem is studied in (Lynce and Ouaknine 2006; Ansótegui et al. 2006). In particular, they give the percentage of problems (from a database of sudoku problems) that can be solved with unit propagation and other forms of restricted inference rules. One of these rules is the so-called *failed literal rule* (Freeman 1995). It is applied as follows, if after assigning $[x \to b]$, where $b \in \{0, 1\}$, and performing unit propagation we get a conflict, then we assign $[x \to \overline{b}]$. This rule together with the unit clause propagation rule characterizes the set of formulas with space two.

Lemma 11 *A formula Γ can be proved unsatisfiable only using unit propagation and the failed literal rule, if, and only if, $s(\Gamma) \leq 2$.*

PROOF: The proof of the equivalence is similar to the proof of Lemma 4. We only need to notice that the resolution steps corresponding to this restricted form of inference have the following form:



In other words, the failed literal rule corresponds to a resolution step where one of the premises has space one, since it is proved only using unit clause propagation. ■

Since all the sudokus analyzed in (Lynce and Ouaknine 2006), using the so-called extended encoding (i.e. adding some redundant clauses that make the problem easier), can be solved only using unit clause propagation and the failed literal rule, we can conclude that all these (extended) encodings have space two.

Comparison with Backdoors and the Minimum Proof Tree Size

In this section we compare our results about the space of an unsatisfiable formula with the size of the minimum *strong* backdoor of the formula. In what follows, when we say backdoor we mean *strong* backdoor.

Given a sub-solver A, and a unsatisfiable formula Γ a strong backdoor (Williams, Gomes, and Selman 2003, Definition 2.4) is a subset S of the variables such that, for every partial variable assignment $\phi : S \to \{0, 1\}$, the sub-solver concludes unsatisfiability for $\phi(\Sigma)$ in polynomial time.

The first thing that must be noticed is that this definition depends on the given sub-solver. First, we will assume that it only performs unit clause propagation, i.e. we will assume that, if the sub-solver accepts a formula, and determines its unsatisfiability, then this formula has space one. Later we will discuss how to extend the comparison in general. In (Williams, Gomes, and Selman 2003, Theorem 4.1), it is proved that deciding if a formula with a backdoor of size b is satisfiable is in $\mathcal{O}(p(n)\left(\frac{2n}{b^{1/2}}\right)^b)$, where $p(n)$ is a polynomial, related with the performance of the sub-solver. In our case, we will assume that $p(n) = n$, as it is the case for the sub-solvers we are considering.

Compared with the performance obtained with the Beame-Pitassi algorithm (Theorem 6) this complexity is better, since we have $b^{1/2}$ in the denominator. However, for constant backdoors sizes and constant space, we have polynomial complexity in both cases, being the space and the backdoor size the exponent of the polynomial.

The following lemma will help us compare the space of a formula with the size of backdoors and cycle-cutsets.

Lemma 12 *Given a formula Γ, a set of variables S, and a value $k \geq 0$, if for any assignment $\phi : S \to \{0,1\}$ we have $s(\phi(\Gamma)) \leq k$, then $s(\Gamma) \leq k + |S|$.*

PROOF: For each ϕ, we construct a tree-like resolution proof $\phi(\Gamma) \vdash \square$, with space bounded by k. Now, from ϕ, we construct the clause that falsifies the assignment $C_\phi = \bigvee_{\phi(x)=0} x \vee \bigvee_{\phi(y)=1} \overline{y}$. Adding some of the literals of C_ϕ to the clauses of $\phi(\Gamma)$ we can get Γ. Doing the same to the proof $\phi(\Gamma) \vdash \square$, we get a proof $\Gamma \vdash C'_\phi$, where $C'_\phi \subseteq C_\phi$, with the same structure, hence space, as the original proof. Therefore, for each one of the $2^{|S|}$ clauses C built up from the variables of S, we have a subclause $C' \subseteq C$, and a proof $\Gamma \vdash C'$. Now, cutting the variables of S we can construct a proof tree of maximal depth $|S|$ that derives the empty clause from the clauses C'. The composition of these trees results in a tree with space bounded by $k + |S|$ that derives the empty clause from Γ. ■

Lemma 13 *Given a sub-solver A that only accepts formulas with space k, if the subset of variables S is a strong backdoor of the formula Γ, then $s(\Gamma) \leq |S| + k$.*

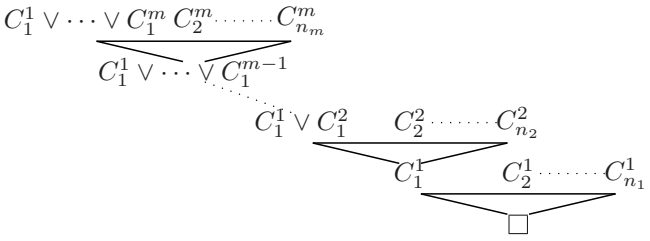
Notice that this lemma concludes that, according to our project goal, the size is a better measure than the minimal backdoor size. However, the result is conditioned by an assumption. If the sub-solvers only accepts Horn clauses, then the result is true with $k = 1$. If the sub-solver works on polynomial time, then it is reasonable to assume that it only accepts formulas with an space bounded by the degree of this polynomial.

On the other hand, the space may be arbitrarily smaller than the size of strong backdoors, as the following example shows.

Example 14 Assume that the following family of unsatisfiable formulas $\Gamma^j = \{C_1^j, \ldots, C_{n_j}^j\}$ are difficult for the sub-solver, for $j = 1, \ldots, m$, and assume that they have disjoint variables. Let b^j be the minimal size of backdoors of Γ^j. Let us define the following unsatisfiable formula

$$\Gamma = \{C_1^1 \vee \cdots \vee C_1^m\} \cup \bigcup_{j=1}^{m} \{C_2^j, \ldots, C_{n_j}^j\}$$

To get an easy formula for the sub-solver we have to instantiate all variables of some backdoor of each subformula, hence the size of minimal backdoors of Γ is $\sum_{j=1}^{m} b^j$. However, the space of Γ is bounded by $\max_{j=1}^{m}\{s(\Gamma^j)\} + 1$. To prove the statement, we can construct the following refutation tree.



The backdoor size satisfies the following property: for any formula Γ, there exists a variable x, such that for any assignment $\phi : \{x\} \to \{0,1\}$, we have $backdoorsize(\phi(\Gamma)) \leq backdoorsize(\Gamma) - 1$. Therefore, the backdoor size decreases in both subtrees of the proof. Notice that this property is similar to property (3) of lemma 3, but more restrictive. This explains why it results into a bigger measure.

As we have already said, in very basic SAT solvers, the *proof tree* and the *search tree* are similar. In the DPLL algorithm, for instance, both trees are equal except for the unit propagation. This means that the Strahler of the search tree can be one unit smaller than the Strahler of the proof tree[4]. The bigger the inference in each node is, the bigger is this difference between the Strahlers of the proof and the search trees. We can define the *practical space* as the Strahler of the *search* tree. Moreover, like for backdoors, we can make this definition parametric on the sub-solver used in the leaves of the search tree. These subsolvers on the leaves would play the same role as the oracles in (Kullmann 1999). In fact, in the experiments of the last section, we estimate the space as the Strahler of the search tree. Then, since the size of the backdoor roughly corresponds to the hight of the search tree, and this is always bigger than its Strahler (specially in unbalanced trees), we can conclude that the practical space is smaller than the backdoor size.

According to our project, the logarithm of the minimum tree-like resolution proof of a formula, noted $\log ts(\Gamma)$ is a *good*, and also natural, measure of its hardness. This is because there exists an algorithm (the original Beame-Pitassi algorithm) that decides satisfiability in time $\mathcal{O}(n^{\log ts(\Gamma)})$. However, according to this project, the space proposed in our paper is a *better* measure, because for any formula Γ:

$$s(\Gamma) \leq \log ts(\Gamma) \leq s(\Gamma) \cdot \log n$$

where n is the number of variables of Γ.

The first inequality holds because the Strahler of a tree is equal to the logarithm of the size of the the maximal complete tree that can be embedded into the tree (see Definition 1), hence smaller that the logarithm of the size of the tree. The second inequality comes from the fact that $ts(\Gamma)$ is smaller than the size of the tree computed by Beame-Pitassi algorithm, and this is bounded by $n^{s(\Gamma)+1}$. For small space instances the $log(n)$ factor is important. For instance, for Horn formulas, the space is always 1, whereas the size of the minimal tree-like proof can be n.

Comparison with Cycle-CutSets and Treewidth

The cycle-cutset technique consists in breaking cycles in graphs of variable dependencies (here, two variables are dependent if they share the same clause) by instantiating some of the variables. A cycle-cutset is a set of variables that after instantiation transform the dependency graph into a tree.

[4]For instance, for Horn formulas, the Strahler of the proof tree is one, whereas the Strahler of the search tree –there is only one node– is zero.

Once we have a tree of dependencies, we can apply simple methods to get the instances of the rest of variables. Therefore, the size of minimal cycle-cutsets gives an idea of how hard a problem is, in fact the method has a time complexity $O(n\,2^c)$, where n is the number of variables, and c the cycle-cutset size. However, we can prove that the space is smaller than the size of the cycle-cutset:

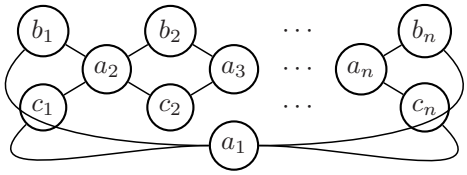
Lemma 15 *For any formula Γ, with a cycle-cutset S we have $s(\Gamma) \leq |S| + 1$.*

PROOF: Sketch: The proof is similar to Lemma 13. After instantiating the cycle-cutset variables, we get a formula with a dependency graph with tree shape. This formula is Horn renamable, hence its space is at most one. ■

Example 16 *The following unsatisfiable formula*

$$\left\{ \begin{array}{l} \overline{a_i} \vee b_i \\ \overline{b_i} \vee a_{i+1} \\ \overline{a_{i+1}} \vee c_i \\ \overline{c_i} \vee a_i \end{array} \right\}_{i=1,\ldots,n-1} \cup \left\{ \begin{array}{l} \overline{a_n} \vee b_n \\ \overline{b_n} \vee \overline{a_1} \\ a_1 \vee c_n \\ \overline{c_n} \vee a_n \end{array} \right\}$$

has space two (since it is a 2-CNF). However, its graph of variable dependencies contains n independent cycles. Therefore, the cycle-cutset size is n.



The previous lemma and example show that, for SAT as a particular case of CSP, the space of a formula is always smaller than the cycle-cutset size, and can be arbitrarily smaller. Therefore the space is a better measure.

In the tree decomposition method (Dechter and Pearl 1989), we put some variables into clusters of variables and structure those clusters as a tree satisfying (1) for every clause, all its variables are inside some of the clusters, and (2) for every variable, the set of clusters containing this variable induces a (connected) subtree. With this method we get a complexity $\mathcal{O}(n\,4^{w\,\log n})$, where n is the number of variables, and w is the treewidth of the decomposition (i.e. the size of the biggest cluster). As it is proved in (Gottlob, Leone, and Scarcello 2000), this method is *better* than the cycle-cutset method. However, like cycle-cutsets, it characterizes tractability due to the structure, not to the constraint relation. This makes easy to find, similarly to Example 16, an unsatisfiable set of clauses where every variable is related with every variable by a binary clause. This formula has space 2, but the treewidth is equal to the number of variables.

Experimental Results

We have conducted a set of experiments on random and industrial SAT instances, with a modified version of the SAT solver satz (Li and Anbulagan 1997) that we will call strahler-satz. For unsatisfiable instances strahler-satz reports

#vars (n)	space (s)	$100 \cdot s/n$
100	5	5
150	6.5	4.33
200	8.1	4.05
250	9.77	3.90
300	11.28	3.76
350	12.8	3.67
400	14.62	3.65

Table 2: Space at the peak in the phase transition region of random 3-SAT instances, ratio $\#clauses/\#vars = 4.25$ (with 20 instances per point).

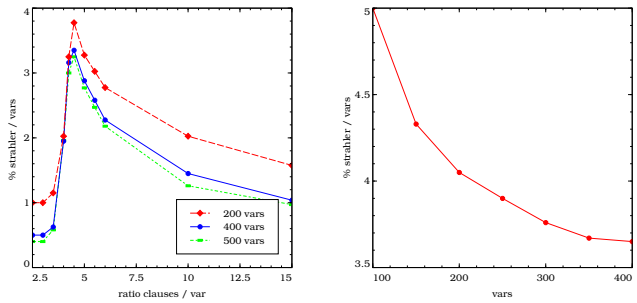


Figure 2: (Left) Evolution of the ration $100 \cdot space/\#vars$ with respect to $\#clauses/\#vars$ for random 3-SAT instances.
(Right) Evolution of the quotient $100 \cdot space/\#vars$ with respect to $\#vars$ for random 3-SAT instances with fixed ratio $\#clauses/\#vars = 4.25$.

the Strahler of the proof tree generated by satz. Since the space of a formula is defined as the minimum Strahler of a proof tree, strahler-satz gives an upper bound on the space (we have checked the program on instances with known space and saw that the upper bound almost coincides with the real value). For satisfiable instances strahler-satz computes the Strahler of the partial proof tree constructed until the satisfying assignment is found. This value is, in general, smaller than s^b defined for satisfiable instances, because s^b corresponds to the Strahler of the proof search tree that needs to be generated to find *all* the satisfying assignments, not just the first one as strahler-satz does. The effect of this underestimation in the computation of the space of satisfiable instances is a shift of the peak of the phase transition to the right.

Table 1 shows the evolution of the space across the phase transition for random 3-SAT instances. As we can see the maximum value of the space is a bit more to the right of the phase transition point 4.25. We conjecture that the *real* (theoretical) space has a peak exactly in the phase transition point. In this table we also show the space filtering satisfiable instances (with an underestimated computation of their space) to show that, for unsatisfiable instances, the space is bigger at 4.25 than at 4.5.

Table 2 shows the evolution of the space for random 3-SAT instances generated at the peak in the phase transition

		#clauses/#vars										
#vars		2.5	3	3.5	4	4.25	4.5	5	5.5	6	10	15
200	sat & unsat	2	2	2.3	4.05	**6.5**	7.55	6.55	6.05	5.55	4.05	3.15
	only unsat	-	-	-	-	**8.1**	7.55	6.55	6.05	5.55	4.05	3.15
400	sat & unsat	2	2	2.5	7.8	**12.65**	13.4	11.55	10.35	9.1	5.8	4.15
	only unsat	-	-	-	-	**14.62**	13.4	11.55	10.35	9.1	5.8	4.15
500	sat & unsat	2	2	2.9	9.8	**15**	16.25	13.85	12.35	10.9	6.3	4.85
	only unsat	-	-	-	-	**17.8**	16.25	13.85	12.35	10.9	6.3	4.85

Table 1: Space across the phase transition region for random 3-SAT instances (20 instances per point).

instance	unsat/sat	#vars(n)	space (s)	$100 \cdot s/n$
sat solver competition 2005				
vmpc27	sat	729	5	0.68
vmpc30	sat	900	4	0.44
depots3_ks99i	sat	1037	7	0.67
driverlog2_v0li	sat	763	6	0.81
ferry6_ks99i	sat	1425	13	0.91
ferry6_ks99a	sat	701	4	0.57
ferry7_ks99a	sat	960	11	1.14
satellite2_v0li	sat	853	8	0.93
www.satlib.org				
ssa2670-141.cnf	unsat	986	16	1.62
ssa2670-130.cnf	unsat	1359	15	1.10
ssa0432-003.cnf	unsat	435	7	1.6
bf2670-001.cnf	unsat	1393	6	0.43
bf1355-075.cnf	unsat	2180	7	0.32
bf0432-007.cnf	unsat	1040	6	0.57
bf1355-638.cnf	unsat	2177	7	0.32

Table 3: Space at industrial instances.

region, ratio 4.25. We can see that the Strahler grows proportionally to the number of variables in the SAT instances. In other words, the quotient $space/\#vars$ seems to tend to 0.036 for big random instances at the phase transition region (see Fig. 2 right).

Finally, Table 3 shows the space for some satisfiable and unsatisfiable industrial instances. As we can observe the ratio $space/\#vars$ is smaller for the industrial instances than for random instances generated at the peak of the phase transition region.

References

Ansótegui, C.; Béjar, R.; Fernández, C.; Gomes, C. P.; and Mateu, C. 2006. The impact of balancing on problem hardness in a highly structured domain. In *AAAI'06*.

Beame, P., and Pitassi, T. 1996. Simplified and improved resolution lower bounds. In *FOCS'96*, 274–282.

Beame, P.; Kautz, H. A.; and Sabharwal, A. 2004. Towards understanding and harnessing the potential of clause learning. *J. of Art. Int. Research* 22:319–351.

Ben-Sasson, E., and Galesi, N. 2003. Space complexity of random formulae in resolution. *Random Struct. Algorithms* 23(1):92–109.

Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Art. Int.* 38(3):353–366.

Dilkina, B. N.; Gomes, C. P.; and Sabharwal, A. 2007. Tradeoffs in the complexity of backdoor detection. In *CP'07*, 256–270.

Ershov, A. P. 1958. On programming of arithmetic operations. *Com. of the ACM* 1(8):3–6.

Esteban, J. L., and Torán, J. 1999. Space bounds for resolution. In *STACS'99*, 551–560.

Esteban, J. L., and Torán, J. 2001. Space bounds for resolution. *Inf. and Comp.* 171(1):84–97.

Freeman, J. W. 1995. *Improvements to Propositional Satisfiability Search Algorithms*. Ph.D. Dissertation, Univ. of Pennsylvania.

Gottlob, G.; Leone, N.; and Scarcello, F. 2000. A comparison of structural CSP decomposition methods. *Art. Int.* 124(2):243–282.

Henschen, L. J., and Wos, L. 1974. Unit refutations and Horn sets. *J. of the ACM* 21(4):590–605.

Horton, R. E. 1945. Erosioned development of streams and their drainage basins, hydrophysical approach to quantitative morphology. *Bull. Geol. Soc. of America* 56:275–370.

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

Kullmann, O. 1999. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. *ECCC* 41.

Kullmann, O. 2004. Upper and lower bounds on the complexity of generalised resolution and generalised constraint satisfaction problems. *Annals of Math. and Art. Int.* 40(3-4):303–352.

Li, C. M., and Anbulagan. 1997. Look-ahead versus look-back for satisfiability problems. In *CP'97*, 341–355.

Lynce, I., and Ouaknine, J. 2006. Sudoku as a SAT problem. In *AIMATH'06*.

Nordström, J. 2006. Narrow proofs may be spacious: separating space and width in resolution. In *STOC'06*, 507–516.

Strahler, A. N. 1952. Hypsometric (area-altitude) analysis of erosonal topology. *Bull. Geol. Soc. of America* 63:1117–1142.

Torán, J. 1999. Lower bounds for space in resolution. In *CSL'99*, 362–373.

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