Relating Proof Complexity Measures and Practical Hardness of SAT

Matti Järvisalo[1], Arie Matsliah[2], Jakob Nordström[3], and Stanislav Živný[4]

[1] Department of Computer Science & HIIT, University of Helsinki, Finland
[2] IBM Research and Technion, Haifa, Israel
[3] KTH Royal Institute of Technology, Stockholm, Sweden
[4] University of Oxford, United Kingdom

Abstract. Boolean satisfiability (SAT) solvers have improved enormously in performance over the last 10–15 years and are today an indispensable tool for solving a wide range of computational problems. However, our understanding of what makes SAT instances hard or easy in practice is still quite limited. A recent line of research in proof complexity has studied theoretical complexity measures such as *length*, *width*, and *space* in resolution, which is a proof system closely related to state-of-the-art conflict-driven clause learning (CDCL) SAT solvers. Although it seems like a natural question whether these complexity measures could be relevant for understanding the practical hardness of SAT instances, to date there has been very limited research on such possible connections. This paper sets out on a systematic study of the interconnections between theoretical complexity and practical SAT solver performance. Our main focus is on space complexity in resolution, and we report results from extensive experiments aimed at understanding to what extent this measure is correlated with hardness in practice. Our conclusion from the empirical data is that the resolution space complexity of a formula would seem to be a more fine-grained indicator of whether the formula is hard or easy than the length or width needed in a resolution proof. On the theory side, we prove a separation of general and tree-like resolution space, where the latter has been proposed before as a measure of practical hardness, and also show connections between resolution space and backdoor sets.

1 Introduction

In the last 10–15 years, SAT solvers have become a standard tool for solving a wide variety of real-world computational problems [1]. Although all known SAT solvers have exponential running time in the worst case, dramatic improvements in performance have led to modern SAT solvers that can handle formulas with millions of variables. At the same time, very small formulas with just a few hundred variables are known which are completely beyond the reach of even the very best solvers. Understanding what makes a SAT instance hard or easy for state-of-the-art SAT solvers is therefore a fundamental problem. In particular, a natural, but not at all well-understood, question is whether one can find a good *measure on the practical hardness* of SAT instances.

The current work addresses this question from the viewpoint of *conflict-driven clause learning* (CDCL) solvers [2, 3, 4], which—applying efficient data structures,

clause learning, forgetting, restarting, phase saving, and other important schemes—form the most prominent SAT solver paradigm today. Our goal is to explore possible connections between *practical hardness*—as witnessed by running times of CDCL solvers on SAT instances—and *proof complexity measures* employed in the formal study of the resolution proof systems that can be seen to underlie CDCL SAT solvers.

The main bottleneck for CDCL solvers—apart from the obvious exponential worst case behavior—is the amount of memory used. In practice, it is completely infeasible to store all clauses learned during a CDCL run, and one therefore needs to design a highly selective and efficient clause caching scheme that learns and keeps the clauses needed for the CDCL solver to finish fast. Thus, understanding time and memory requirements for clause learning algorithms, and how these resources are related to each other, is a question of great practical importance.

Proof complexity provides a possible approach for analyzing the potential and limitations of SAT solvers by studying the formal systems of reasoning which the solvers use to generate proofs. A lower bound for a proof system tells us that any algorithm, even an optimal (non-deterministic) one making all the right choices, must necessarily use at least the amount of a certain resource specified by this bound. In the other direction, theoretical upper bounds on a proof complexity measure give hope that SAT solvers can perform well with respect to the measure if an efficient search algorithm can be designed. Whereas the plain Davis–Putnam–Logemann–Loveland procedure (DPLL) [5, 6] is known to correspond to *tree-like* resolution, by recent theoretical accounts (including [7, 8]) CDCL solvers can be understood as deterministic instantiations of the general resolution proof system. In this context, the proof complexity measures of *length* (a.k.a. size) and *space* are interesting since they in some sense model the running time and memory consumption of (optimal) CDCL solvers, while *width* is another measure that seems relevant for practical performance in view of e.g. [9]. An informal description of these measures is as follows (see Section 2 for the formal definitions):

Length: The number of clauses in a resolution proof.
Width: The size of a largest clause in a resolution proof.
Space: The number of clauses needed "on the board" in a self-contained presentation of a proof, where inferences can only be made from what is currently on the board.

The length, width, or space of proving a formula is defined in the natural way by taking the minimum over all possible proofs with respect to the measure in question. As will be discussed in more detail below, these measures have been proven to form a strict hierarchy in the sense that for any formula we have the (informally stated) relations

$$\text{space} \geq \text{width} \geq \log \text{length} \tag{1}$$

(scaling length so that all measures have the same magnitude—length can be exponential while space and width are always at most linear), and these inequalities can all be asymptotically strict.

When discussing whether these theoretical complexity measures have any bearing on the practical hardness of formulas, perhaps the most obvious candidate to start with is *length*. Indeed, if the shortest resolution proof is very long then clearly no CDCL solver can do well, since a resolution proof can be extracted from the run of such a solver.

Thus a strong lower bound on length is a sure indicator of hardness. But in the opposite direction, the fact that there exists a short resolution proof does not mean that this proof is easy to find. On the contrary, [10] indicates that it might be computationally infeasible to find any proof in reasonable length even given the guarantee that a short proof exists. Therefore, length seems too "optimistic" as a measure of hardness in practice.

Turning next to *width*, if the shortest proof for a formula is very long, then the width of a "narrowest" proof must by necessity also be large (simply by counting the number of possible clauses). However, the fact that a resolution proof needs to be wide, i.e., has to contain some large clause, does *not* necessarily imply that the minimum length is large [11]. Width is thus a stricter hardness measure than length. Recently, is was shown in [9] (under some theoretical assumptions) that if the minimum width of any resolution proof for a formula is w, then a CDCL solver (that does not at all care about width per se) will with high probability decide the formula in time something like n^w. This seems to indicate that resolution width could be a measure that correlates well with practical hardness for CDCL solvers. On the other hand, some of the technical assumptions needed to establish this result seem somewhat idealized compared to how CDCL solvers work in practice.

Since, as already discussed, memory consumption is a major concern in practice, the related theoretical measure of *space* clearly seems interesting to study in this context. Indeed, this was arguably the main reason why research into proof space complexity was initiated in the late 1990s. It was shown in [12] that if there is no narrow proof for a formula, then there is no small-space proof either. However, large space does *not* imply anything about the width [13], and hence space is an even stricter hardness measure than width. Intuitively, one could argue that for a formula with high space complexity, CDCL solvers would need to learn many clauses and keep them in memory, and therefore such formulas should be hard in practice. A stronger conjecture would be that it also holds that if a formula has low space complexity—which also guarantees that there are both short and narrow proofs—then a CDCL solver should (at least in principle) be able to learn the few clauses needed to quickly produce a short proof.

The purpose of our work is to provide an empirical evaluation of these proof complexity measures and their relevance for hardness in practice, focusing in particular on space. Our work can be seen to implement a program outlined previously in [14]. It should be noted, however, that [14] suggested *tree-like space*, i.e., space measured in the subsystem of resolution producing only tree-like proofs, as a measure of practical hardness. We will return to the question of general versus tree-like space later, but let us just remark here that in tree-like resolution space is tightly correlated with length in the sense that there is a short tree-like proof if and only if there is a space-efficient one [15] (which is provably not true in general resolution by [13]). If tree-like space were a good indicator of hardness in practice, this would mean that CDCL solvers could decide a formula efficiently if and only if it had a short *tree-like* proof, which in turn would seem to imply that in practice CDCL solvers cannot provide any significant improvements in performance over plain DPLL solvers. This does not seem consistent with observations that the former can vastly outperform the latter; observations that have been corroborated by theoretical papers arguing that CDCL proof search is closely related to *general* resolution [7, 8] instead of mere *tree-like* resolution.

1.1 Contributions of This Paper

The main contribution of this work is to provide a first extensive empirical evaluation of the connections between resolution space complexity and practical hardness for CDCL solvers. We remark that although [14] suggested this, as far as we are aware no such evaluation has previously been carried out (and, indeed, has only been made possible by recent advances in proof complexity providing formulas with the necessary theoretical guarantees). We investigate to what extent resolution space correlates with running times for state-of-the-art CDCL solvers. In the experiments, we employ highly structured SAT instances—so-called *pebbling formulas*—allowing us to make controlled observations on the relation between space complexity and SAT solver performance. Pebbling formulas are guaranteed to be very easy with respect to length (the short proof consists of just listing the clauses of the formula in the right order and doing a very small constant number of intermediate derivation steps in between each such clause) and also with respect to width (the proofs just sketched will have width equal to the width of the formula, which in all cases is a one-digit constant) but can be varied with respect to their space complexity from constant all the way up to almost linear. This makes it possible to study which level of granularity (if any) is the best one in the hierarchy of measures in (1) when we are looking for indicators of hardness in practice. If length or width were the most relevant measure, then CDCL solvers should behave in essentially the same way for all formulas in our experiments. If, however, the more fine-grained measure of space is a more precise indicator of hardness, then we would expect to see a clear correlation between running times and theoretical space complexity. As we will argue below, the conclusion from our experiments is that the latter case seems to hold.

Complementing the empirical evaluation, we also present some theoretical results related to resolution space. In particular, we prove the first non-trivial separation of general and tree-like resolution space. Previously, only a constant-factor separation was known [16], meaning that it could not be ruled out that the two measures were essentially the same except for small multiplicative constants. We improve this to a logarithmic separation, which, while still leaving room for further improvements, shows that the measures are fundamentally different. This in turn motivates our focus on general space as a measure of practical hardness for CDCL solvers. Furthermore, elaborating on and extending related results in [14], we also address the relation between resolution space and *backdoor sets* [17], a somewhat more practically motivated measure previously proposed as a proxy for practical hardness.

2 Proof Complexity Preliminaries

We now give a brief overview of the relevant proof complexity background; for more details, see e.g. [18]. We assume familiarity with CNF formulas, which are conjunctions of clauses, where a clause is a disjunction of literals (unnegated or negated variables, with negation denoted by overbar). It is convenient to view clauses as sets, so that there is no repetition of literals and order is irrelevant. A k-CNF formula has all clauses of size at most k, which is implicitly assumed to be some (small, say one-digit) constant throughout this paper. Below we will focus on k-CNF formulas to get cleaner statements of the theoretical results (analogous results hold in general but are not as simple to state).

A *resolution refutation* $\pi\colon F \vdash \bot$ of an unsatisfiable CNF formula F, also known as a *resolution proof* for F, is an ordered sequence of clauses $\pi = (D_1, \ldots, D_\tau)$ such that $D_\tau = \bot$ is the empty clause containing no literals, and each line D_i, $1 \leq i \leq \tau$, is either one of the clauses in F (*axioms*) or is derived from clauses D_j, D_k in π with $j, k < i$ by the *resolution rule* $\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$ (where the clause $B \vee C$ is the *resolvent* of the clauses $B \vee x$ and $C \vee \overline{x}$ on x). With every resolution proof π we can associate a graph G_π by having a sequence of vertices v_i labelled by the clauses D_i on a line in order of increasing i, and with edges from v_j and v_k to v_i if D_i was derived by resolution from D_j and D_k. Note that there might be several occurrences of a clause D in the proof π, and if so each occurrence gets its own vertex in G_π.

The *length* $L(\pi)$ of a resolution proof π is the number of clauses in it (counted with repetitions). The *width* $W(C)$ of a clause C is $|C|$, i.e., the number of literals, and the width $W(\pi)$ of a proof π is the size of a largest clause in π. The *space* (sometimes referred to as *clause space*) of a proof at step i is the number of clauses C_j, $j < i$, with edges to clauses C_k, $k \geq i$, plus 1 for the clause C_i derived at this step. That is, intuitively space measures the number of clauses we need to keep in memory at step i, since they were derived before step i but will be used to infer new clauses after step i (or possibly at step i). The space of a proof is the maximum space over all steps in the proof. *Tree-like (clause) space* is defined in exactly the same way except that the graph G_π representing π is constrained to be a (binary) tree.

We next briefly review what is known about these measures. As shown in [19, 20] and many later papers, the length of refuting a CNF formula F can be exponential in the size of F (measured as the total number of literals counted with repetitions), and it is easy to show that the worst case is at most exponential. For width, clearly the size of the formula (and, in particular, the number of distinct variables in it) is an upper bound, and there are matching lower bounds up to constant factors [21]. If a formula has a narrow proof then this proof must also be short (simply by counting the total number of distinct clauses). The opposite does not necessarily hold as proven in [11] (although very strong lower bounds on width do imply strong lower bounds on length by [21]).

Just as for width, although somewhat less obviously, space is also at most linear (even for tree-like space) as shown in [15], and again there are matching lower bounds, e.g., in [22, 23]. In [12] it was shown that if a formula can be refuted in small space, this implies there is also a small-width proof (although in general this will not be the same proof). The converse of this is false in the strongest sense possible—there are formulas with constant-width proofs that require almost linear (i.e., worst-case) space [13]. Since space upper-bounds width, and also width upper-bounds length as discussed above, it follows that upper bounds on space imply upper bounds on length. Conversely, in [15] it was shown that small length implies small space for the restricted case of tree-like resolution. In general resolution, however, the fact that a formula is refutable in small length says essentially nothing about the space complexity [13].

3 Pebbling Formulas

To study the proof complexity measures of length, width and space, and to relate them to the practical hardness of CNF formulas, we focus on so-called *pebbling formulas* (also

known as *pebbling contradictions*). Our main motivation for using pebbling formulas is that, as explained below, recent theoretical advances allow us to construct such formulas with varying (and fully specified) space complexity properties while keeping the length and width complexity fixed.

Pebbling formulas are so called since they encode instances of *pebble games* played on directed acyclic graphs (DAGs). Wee refer to the survey [24] for more information about such games. The pebbling formula over G associates one variable with each vertex, postulates the source vertices (with no incoming edges) to be true and the (unique) sink vertex (with no outgoing edges) to be false, and then specifies that truth propagates from the sources to the sink. More formally, as defined in [21] the pebbling formula Peb_G over a DAG G consists of:

- for all source vertices s in G, a unit clause s (*source axioms*),
- for all non-sources v with incoming edges from the vertex set $pred(v)$ of immediate predecessors of v, the clause $\bigvee_{u \in pred(v)} \overline{u} \vee v$ (*pebbling axioms*),
- for the (unique) sink z of G, the unit clause $\overline{z}$ (*sink axiom*).

If G has n vertices and max fan-in ℓ, then Peb_G is an unsatisfiable $(1{+}\ell)$-CNF formula with $n+1$ clauses over n variables. For all graphs used in this paper we have $\ell = 2$.

Pebbling formulas are not of much use to us as such—they are very easy with respect to all proof complexity measure we have discussed, and are easily seen to be solvable simply by unit propagation. However, they can be transformed into much more interesting formulas by substituting Boolean functions for the variables as follows.

Given any CNF formula F, we can fix a Boolean function $f\colon \{0,1\}^d \mapsto \{0,1\}$ and substitute every variable x in F by $f(x_1, \ldots, x_d)$, where $x_1, \ldots, x_d$ are new variables that do not appear anywhere else. Then we expand this out to get an equivalent CNF formula over this new set of variables. For a small example, if we let $\oplus$ denote binary exclusive or, then the clause $\overline{x} \vee y$ after substitution becomes $\neg(x_1 \oplus x_2) \vee (y_1 \oplus y_2)$ which is expanded out to the set of clauses

$$\{x_1 \vee \overline{x}_2 \vee y_1 \vee y_2,\ x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2,\ \overline{x}_1 \vee x_2 \vee y_1 \vee y_2,\ \overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2\}\ . \quad (2)$$

(For general f there might be some choices in exactly how to do this expansion, but such implementation details do not affect this discussion so we ignore them for brevity.)

The *pebbling price* of a DAG G measures how much space is needed to pebble G. As shown in [13, 25], making substitutions in pebbling formulas using *robust* functions f, meaning that the truth value of f can never be fixed by just assigning to one variable, yields substituted CNF formulas for which the *space complexity of the formula coincides with the pebbling price of* G.[5] The canonical example of a robust function is exclusive or. A non-robust function is ordinary or, but [26, 27] show that even for this function the same connection holds at least for certain fairly general families of graphs. This means that if we pick the right graphs, we can generate CNF formulas with known

[5] Actually, this is an oversimplification and formally speaking not correct—the space will be somewhere in between the deterministic black and the (smaller) non-deterministic black-white pebbling price, but these two measures are within a small constant factor for all graphs considered in this paper so this is immaterial. We emphasize that all constants involved are explicitly known and are very small, with the single exception of the gtb graphs discussed below.

Table 1. DAG families and properties of the resulting CNF formula families

Name	Description	Space cplx
pyr$\langle h \rangle$seq	Sequence of pyramid graphs of (constant) height h	$\Theta(h)$
width$\langle w \rangle$chain	Chain graph of (constant) width w	$\Theta(w)$
bintree	Complete binary tree	$\Theta(\log n)$
pyrofpyr	Pyramid of height $\sqrt[4]{n}$ with each node expanded to pyramid	$\Theta(\sqrt[4]{n})$
pyrseqsqrt	Sequence of pyramids of (growing) height $\sqrt[4]{n}$	$\Theta(\sqrt[4]{n})$
pyramid	Pyramid graph of (growing) height $\sqrt{n}$	$\Theta(\sqrt{n})$
gtb	DAGs from [28] with butterfly graphs as superconcentrators	$\Theta(n/\log^2 n)$

space complexity. In addition, it is easy to show that any pebbling formula, even after substitution, can be refuted in (small) constant width and (small) linear length.[6] Thus, in this way we can get formulas that are uniformly *very* easy with respect to length and width, but for which the space complexity varies.

Such formulas would seem like excellent benchmarks for testing the correlation between theoretical complexity measures and hardness in practice, and in particular for investigating at which level of granularity theoretical hardness should be measured given the hierarchy in (1). If the minimum width, or length, of a proof for a formula F were good indicators of whether F is hard or easy, then we would expect to get similar running times for pebbling formulas over all graphs of the same size (when fixing the substitution function). If the more fine-grained measure of space is a more precise indicator, however, we would expect running time to correlate with space complexity. Carrying out large-scale experiments along these lines and analyzing the results is the main practical contribution of this paper. When designing such experiments, one needs to choose (a) graphs from which to generate the benchmarks, and (b) substitution functions to apply. We discuss this next.

An overview of our choice of graph families and their space complexities is given in Table 1. Let us first explain two important building blocks. A *pyramid* of height h is a layered DAG with $h+1$ layers, where there is one vertex in the highest layer, two vertices in the next layer, et cetera, down to $h+1$ vertices in the lowest layer, and where the ith vertex at layer L has incoming edges from the ith and $(i+1)$st vertices at layer $L-1$. A *chain* of width w is a layered graph with w vertices at each layer, and with vertices i and $i-1$ at layer $L-1$ having edges to vertex i in layer $L \pmod{w}$.

To obtain two different types of graphs of constant space complexity, we consider sequences of pyramids of constant height h with the sink of each pyramid connected to the leftmost source of next pyramid (pyr$\langle h \rangle$seq) and chains of constant width w (width$\langle w \rangle$chain). Another graph family that should yield easy formulas are complete binary trees (bintree), the space complexity of which is equal to the height of the tree.

To get "medium-hard" DAGs, we use pyramids in two different ways. In pyramid-of-pyramids graphs (pyrofpyr) we take a pyramid of height h and expand each of its

[6] We will not elaborate on exact constants due to space constraints, but all k-CNF formulas considered have $4 \leq k \leq 9$ and the refutation width coincides with the formula width. As to length, a pebbling formula generated with binary XOR substitution and having L clauses is refutable in length $2.25 \cdot L$, and the blow-up for other substitution functions is similar.

Table 2. Substitution functions

Name	Description	Output	CNF encoding (for $d = 3$ variables)
or_d	OR of d vars	$x_1 \vee \cdots \vee x_d$	$x_1 \vee x_2 \vee x_3$
xor_d	parity of d vars	$x_1 \oplus \cdots \oplus x_d$	$x_1 \vee x_2 \vee x_3$, $x_i \bigvee_{j \neq i} \overline{x}_j$, $i = 1, 2, 3$
maj_d	majority of d vars	$2(x_1 + \cdots + x_d) > d$	$x_1 \vee x_2$, $x_1 \vee x_3$, $x_2 \vee x_3$
eq_d	all d vars equal	$x_1 = \cdots = x_d$	$x_1 \vee \overline{x}_2$, $\overline{x}_1 \vee x_2$, $x_1 \vee \overline{x}_3$, $\overline{x}_1 \vee x_3$
e1_d	exactly one of d	$x_1 + \cdots + x_d = 1$	$x_1 \vee x_2 \vee x_3$, $\overline{x}_1 \vee \overline{x}_2$, $\overline{x}_1 \vee \overline{x}_3$, $\overline{x}_2 \vee \overline{x}_3$
s_id	if-then-else	$x_1 ? x_2 : x_3$	$\overline{x}_1 \vee x_2$, $x_1 \vee x_3$

vertices v to pyramid of same height h with sink z_v. Every incoming edge to v is drawn to all sources of the pyramid, and all outgoing vertices from v are drawn from z_v. It is an easy argument that the space complexity is roughly $2h$ and the size of the graph is roughly h^4, so the space complexity grows like $\sqrt[4]{n}$ for graphs of size n. Another way of getting graphs of the same space complexity is to use the same construction as in pyr$\langle h \rangle$seq above but employ graphs of height $\sqrt[4]{n}$. These are our pyrseqsqrt graphs.

Finally to get "really hard" graphs we consider two well-known graph families. The first one is simply pyramids of height (and hence space complexity) $\sqrt{n}$. The second is based on DAGs with maximal space complexity $\Theta(n/\log n)$ [28]. These graphs cannot be used as-is, however. The bound in [28] is asymptotic, with the smallest instances of huge size (due to the need for so-called *superconcentrators* of linear size). Therefore, we modify the construction to use much simpler, but asymptotically worse, superconcentrators made from butterfly graphs. It is not hard to verify that the proofs in [28] still go through, and we get much smaller graphs (gtb) that we can actually use to generate CNF formulas, at the price of paying a log factor in the space complexity.

When choosing the substitution functions to apply for pebbling formulas generated from graphs in these families, we want to achieve two objectives. On the one hand, we would like the functions to be robust (as explained above). On the other hand, however, we do not want too large a blow-up in formula size when substituting functions for variables. Again due to space constraints, we cannot go into too much details, but Table 2 presents our choice of substitution functions, which seem to provide a good trade-off between the two goals, and describes their CNF encodings. Note that we also use the (non-robust) standard non-exclusive or functions, which nevertheless provably preserves the space complexity (albeit with worse guarantees for the hidden constants) for all graph families considered here except the gtb family.

4 Improved Separation of General and Tree-like Resolution Space

Before reporting on the empirical part of this work, we return to the question motivated by [14] of whether tree-like space or general space is likely to be the most relevant space measure when it comes to hardness. We already explained in the introduction the reasons for our skepticism regarding tree-like space as a good hardness measure for CDCL. In this section, we complement this with a more theoretical argument, showing that the tree-like and general resolution space measures are different. Our logarithmic

separation, stated next, improves on the constant-factor separation in [16] which is all that was known previously.

Theorem 1. *There are families of 4-CNF formulas $\{F_n\}_{n=1}^{\infty}$ of size $\Theta(n)$ such that their space complexity in general resolution is $\mathrm{O}(1)$ whereas the tree-like space complexity grows like $\Theta(\log n)$.*

Proof. To prove Theorem 1, we apply the equivalence of the tree-like space of a CNF formula F and Prover-Delayer game on F as described in [16]. The Prover asks about variable assignments, and the delayer answers true, false or $*$ to each query. If the answer is $*$, Prover picks an assignment adversarially but Delayer scores a point. The game ends when Prover has forced a partial truth value assignment that falsifies some clause of F. If Delayer can score exactly p points with an optimal strategy, then the tree-like space complexity is $p + 2$, and the opposite also holds.

Consider a graph that is just a line $(v_1, v_2, \ldots, v_n)$ of length n with edges from each vertex v_i to the next vertex v_{i+1} on the right. Let F_n be the pebbling formula over this graph with substitution by (binary) XOR $\oplus$ as described in Section 3.

It is immediate that the general resolution space complexity is constant. Just start with the leftmost node v_1, for which the XOR of the two associated variables holds in view of the source axioms. Then derive step by step, using pebbling axioms, that if the XOR holds at one vertex v_i, then this implies it also holds at the next vertex v_{i+1}. Finally we reach the rightmost vertex v_n, where the sink axioms say that XOR does not hold. Contradiction.

Now we give a Delayer strategy that scores $\log_2 n$ points. For every vertex, the first time Prover asks about any of the two variables associated to the vertex Delayer answers $*$ and scores. The first time Prover asks about a second variable, Delayer looks whether the vertex v_i is in the leftmost or rightmost half of (the remaining part of) the graph. In the former case, Delayer answers so that the XOR of the two variables is satisfied, which gives a problem instance of the same type of at least half the size over $(v_{i+1}, \ldots, v_n)$. In the latter case, Delayer makes sure the XOR is false. Then Prover has to continue playing in $(v_1, \ldots, v_{i-1})$ to falsify the formula since the rest is now satisfiable. Since Prover can only halve the size of the graph for each second question, and Delayer scores a point for each first question, the tree-like space complexity is $\Omega(\log n)$, and since Prover can use precisely this strategy, the space bound is tight. □

As a final remark, let us note that this proof works equally well for CNF formulas generated from the pyr$\langle h \rangle$seq and width$\langle w \rangle$chain graphs in Section 3.

5 Experimental Evaluation

This section summarizes results of our experiments running state-of-the-art CDCL SAT solvers on pebbling formulas with varying resolution space.[7] As benchmarks, we used

[7] The only experiments previously reported for CDCL solvers on pebbling formulas we are aware of were on "grid formulas", i.e., formulas over pyramids with or_2 using zChaff, with a somewhat different motivation [29].

CNF formulas that we generated for *all* combinations of the graphs mentioned in Table 1 (we used pyr$\langle h \rangle$seq with $h \in \{1, 3, 5, 10\}$ and width$\langle w \rangle$chain with $h \in \{2, 5, 10\}$; overall 12 graph families) and the substitution functions mentioned in Table 2 (we used e1_3, maj_3, or_2, or_3, or_4, s_id, xor_2, eq_3; overall 8 functions) yielding a total of 96 families of CNF formulas. Due to the massive amount of data produced, here we only provide a snapshot of the results. Complete data for all experiments as well as a more detailed description of the formula instances used can be found at `http://www.csc.kth.se/~jakobn/publications/cp12/`.

5.1 Experiment Setup

For the experiments, we used the CDCL solvers Minisat 2.2.0 [30] and Lingeling[8] [31]. We ran the solvers on all CNF families in two modes: "as-is", and with all preprocessing (and all inprocessing for Lingeling) disabled. The experiments were run under Linux on a Intel Core i5-2500 3.3-GHz quad-core CPU with 8 GB of memory. We limited the run-time of each solver to 1 hour per instance.

5.2 Results

First recall that the only parameter that varies among the different formula families is their resolution space complexity. Namely, for each of the 96 families, every formula with n variables has $O(1)$ resolution width and $O(n)$ resolution length, with small hidden constants depending on the substitution function only; only the resolution space varies from $O(1)$ to $\Omega(n/\log^2 n)$ as explained in Section 3.

Overall, the results show a notable difference in running times between different families. In particular, we observed that the hardest families with respect to space complexity are also hardest in practice. In other words, *for these families we observed a clear correlation between space complexity and practical hardness*. While there are some observed exceptions (mainly in the lower-end spectrum of the space complexity), there is a positive correlation between run-times and resolution space for almost all of the families.

An example of the results for both of the solvers with preprocessing turned off is given in Figure 1. For clarity, we only include the following 5 families in each plot (listed in non-increasing order of space complexity, cf. Table 1): pyr1seq, bintree, pyrseqsqrt, pyramid, and gtb.

One reason to run experiments over all combinations of graphs and substitution function is to distill the dependence on space and filter out other factors if possible. For instance, different substitution function can (and will) have different properties in practice although their theoretical guarantees are the same. By aggregating results over all substitution functions instead of just considering one or two functions, we get an overall picture. This is summarised in Table 3, which gives average run-times per instance, normalized by the number of variables and calculated over all considered substitution functions with and without preprocessing. In particular, for each family, we take run-times/size (where size is the number of variables in the formula) for each instance, and

[8] Version 774, with an option to disable pre- and inprocessing, was provided by Armin Biere.

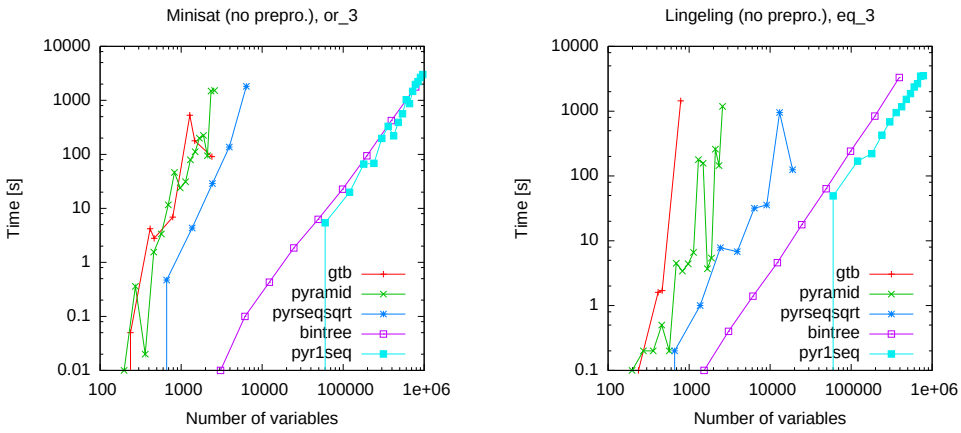

Fig. 1. Results without preprocessing

sum all these numbers for the chosen family and then take an arithmetic mean; that is, $(\text{run-time}_1/\text{size}_1 + \ldots + \text{run-time}_k/\text{size}_k)/k$. The families in Table 3 are *listed in non-increasing order of space complexity* (cf. Table 1). The numbers, which are multiplied by 10^4, show that the average run-times (without preprocessing, first column) correlate strongly with space complexity, the bintree family being the only significant exception. However, bintree graphs have an exceptionally large number (half) of source nodes, which get translated into simpler clauses compared to clauses from non-source nodes. (For a majority of the substitution functions, namely or_2, xor_2, maj_3, e1_d, and s_id, most or all of the resulting clauses are binary.)

Table 3. Average run-times for all families

Time [seconds] $\times$ 10e4		
formula family	**no preprocessing**	**with preprocessing**
gtb	637.65	9.36
pyramid	569.13	1.14
pyrofpyr	234.17	9.45
pyrseqsqrt	122.77	1.64
bintree	12.27	0.27
width10chain	59.87	5.79
pyr10seq	44.12	2.70
width5chain	56.77	5.97
pyr5seq	31.60	2.09
pyr3seq	33.35	1.24
width2chain	45.24	3.25
pyr1seq	39.43	0.36

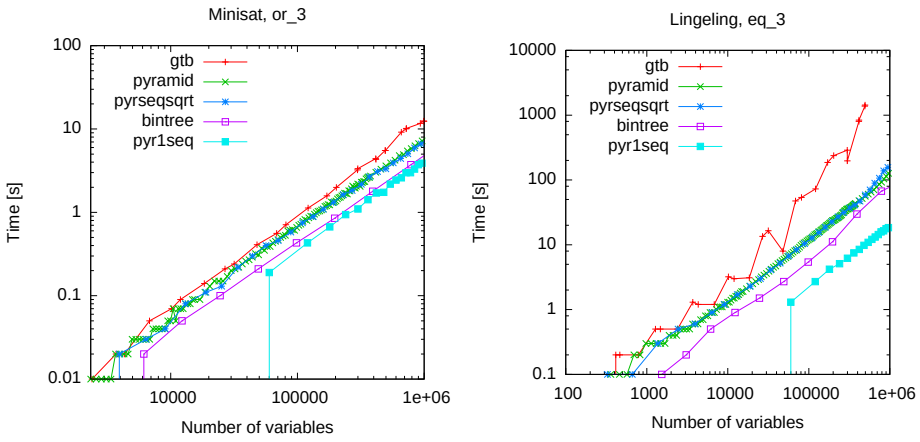

Fig. 2. Results with preprocessing

5.3 Effects of Preprocessing

While a clear correlation between space complexity and solver running times was observed on a variety (but not all) of the formula families, we observed that preprocessing and inprocessing resulted in rather different results. In particular, as exemplified in Figure 2, although smaller correlations can still be observed (especially for Lingeling), preprocessing appears to even out many of the observed differences in the data for the solvers without preprocessing. While this is an observation that clearly sets apart the preprocessing techniques from the behavior of the core CDCL techniques, we do not find it too surprising. Simply, the theoretical space measure that we study can be expected to correlate more or less well with what is going on during clause learning. When preprocessing is applied to a formula F, however, what is fed to the clause learning solver is another formula F' for which we have no theoretical guarantees as to the space complexity (it can a priori be both lower and higher). Furthermore, the fact that our benchmarks have been chosen specifically to be very easy with respect to length and width also means that they are likely to be amenable to the kind of heuristics used in preprocessing. We see much scope for future work here, including ideas how to modify formulas so that they have the same theoretical guarantees but so that these guarantees are more likely to "survive" the preprocessing stage.

5.4 Considerations

We remark that one reasonable objection is that since our benchmarks are pebbling formulas generated from graph it is not clear that we are measuring space complexity per se—maybe we are measuring some other, unrelated graph property. And indeed, some graph properties, such as the number of source nodes, translate into properties of the formulas (many small clauses) that are not captured by the space complexity.

This problem is hard to get around, however. Resolution space complexity seems likely to be PSPACE-complete, and it is an easy argument that is is NP-hard to approximate in any meaningful way, so we cannot expect to be able to determine the space complexity of arbitrary formulas. Instead, we have to pick special instances where we know the space complexity for other reasons, which is the case for the pebbling formula families considered in our experiments.

6 Relating Resolution Space and Backdoors

Compared to the proof complexity hardness measures of resolution length, width, and space, a more practically motivated well-known hardness measure is the size of *(strong) backdoor sets*. Backdoors sets were first studied in [17], and this and subsequent works have shown that real-world SAT instances often have small backdoors, which might offer an explanation why modern SAT solvers perform notably well on such instances.

The definition of backdoor sets is made with respect to a polynomial-time subsolver A. Given a subsolver A and a unsatisfiable formula F, a (strong) backdoor set S is a subset of the variables in F such that for every truth assignment ρ over S, the subsolver A determines unsatisfiability of $F\restriction_\rho$.[9] The algorithm A might be, for instance, unit propagation, polynomial-time restricted DPLL, or a 2-SAT algorithm.

In [14], it was shown that given a subsolver that only accepts formulas with tree-like resolution space k, a CNF formula F, and a strong backdoor set S of F, the tree-like resolution space of F is bounded from above by $|S| + k$. In fact, when restricting to such abstract subsolvers that only accept formulas with tree-like resolution space k, the minimum backdoor set size is a *proper* upper bound for tree-like resolution space. However, [14] does not seem to elaborate too much on what such abstract subsolvers might be. Our following theorem states the relationship between resolution space and backdoor sets in a more concrete way.

Theorem 2. *The following claims hold for any CNF formula F over n variables.*

1. *If F has a backdoor set B of size b with respect to 2-CNF, then the space complexity of F is at most $b + \mathrm{O}(1)$.*
2. *If F has a backdoor set B of size b with respect to unit propagation, then the space complexity of F is at most $b + \mathrm{O}(1)$.*
3. *If F has a backdoor set B of size b with respect to DPLL running in time $\mathrm{poly}(n)$, then the space complexity of F is at most $b + \mathrm{O}(\log n)$.*

Proof. Suppose that ρ is a (partial) truth value assignment to the b variables in the backdoor set B, and that C_ρ is the unique minimal clause falsified by ρ. Then if π_ρ is a resolution refutation of $F\restriction_\rho$ in clause space s, by plugging in F instead of $F\restriction_\rho$ we get from π_ρ a resolution derivation of the clause C_ρ in the same space s. Also, it is easy to show that the set of clauses $\{C_\rho \mid \rho \text{ all total assignments to } B\}$ can be refuted by a (tree-like) resolution refutation π_B in simultaneous space $b + \mathrm{O}(1)$ and length 2^{b+1}. Thus, if each clause C_ρ is derivable in clause space at most s, then we can combine these derivations

[9] $F\restriction_\rho$ denotes F restricted by ρ, i.e., with variables set according to ρ after which the formula is simplified by removing satisfied clauses and unsatisfied literals from clauses.

π_ρ with the refutation π_B of the set of clauses $\{C_\rho \mid \rho \text{ all total assignments to } B\}$ to get a refutation of F in space $b + s + \mathrm{O}(1)$, simply by running π_B, and whenever a clause C_ρ is needed (which will only be once per clause) call on π_ρ as a subroutine.

What remains is to upper bound the space complexity of $F\restriction_\rho$.

1. Any 2-CNF formula can be refuted by resolution in clause space at most 4 [15].
2. Unit propagation can be seen as a resolution proof DAG that is a long chain with every chain vertex having a unique predecessor except for its predecessor on the chain. Such a proof is in (tree-like) clause space 3.
3. If DPLL runs in time n^c, then it produces a tree-like resolution proof in size n^c. According to [15], tree-like resolution length L implies that one can do length L and space $\log L + \mathrm{O}(1)$ simultaneously. Hence we have space $b + \mathrm{O}(\log n)$. □

Recently, the concept of *learning-sensitive backdoors* [32] was proposed as a concept more tightly connected with CDCL solvers than the original definition of backdoors. It was shown that strong learning-sensitive backdoors can be exponentially smaller than traditional backdoors [32]. It remains an interesting open question whether general resolution space is bounded from above by the size of learning-sensitive backdoors, i.e., whether small learning-sensitive backdoors imply low (general) resolution space complexity.

7 Concluding Remarks

This paper advances and expands on the program outlined first in [14], namely, to shed light on possible connections between theoretical complexity measures and practical hardness of SAT, and in particular on whether space complexity is a good indicator of hardness. We provide an extensive empirical evaluation on the correlation between resolution space and practical hardness, running state-of-the-art CDCL SAT solvers on benchmark formulas with theoretically proven properties based on recent results in proof complexity. To the best of our knowledge, no such experiments have previously been done, and we consider this a conceptually important step towards the more general goal of relating complexity of SAT solving in theory and practice. Furthermore, complementing the empirical evaluation, we prove new theoretical results related to resolution space, in particular separating general and tree-like resolution space and thus showing that the two measures are indeed different. We also address the relation between resolution space and backdoor sets.

Regarding the empirical work, while the results presented here do not provide conclusive evidence for resolution space being the "ultimate right measure" of practical hardness (it may be safe to assume that theory and practice most often do not behave exactly identically), the important observation is that we do see nontrivial correlations. We therefore argue that our results are consistent with the hypothesis that resolution space complexity should be a relevant measure of hardness in practice for CNF formulas. In particular, space might be a more precise indicator of practical hardness than length or width, in the sense that the latter two measures give too optimistic estimates for formulas which have very low length or width complexity but which might nevertheless be hard for state-of-the-art CDCL SAT solvers to solve in practice.

We have already discussed at the end of Section 5 why the experiments by necessity had to be run on designed combinatorial benchmark formulas rather than on real-world instances. Another possible issue is that to truly understand the relation between practical hardness on one hand, and length, width and space complexity on the other, one would need experiments that vary all three parameters. However, while this a priori seems like a very reasonable request, the hierarchy between these measures in (1) means that such experiments unfortunately are provably impossible to perform. As soon as the length or width complexity increases, the space complexity will increase as well. Thus, all we can hope to study is whether space provides a more precise indication of hardness than is given by width or length.

We view our work as only a first step in an interesting line of research, and see many important questions to investigate further. One such question closely related to the current paper would be to study the recent theoretical results on trade-offs between proof length and proof space for resolution in [25, 33], and perform experiments on whether these results translate into trade-offs between running time and memory consumption for CDCL solvers in practice.

Acknowledgements

We thank Armin Biere for providing a modified version of Lingeling with pre- and inprocessing switched off, and Niklas Eén and Niklas Sörensson for useful discussions and advice regarding MiniSAT. We also thankfully acknowledge technical assistance at KTH Royal Institute of Technology from Torbjörn Granlund and Mikael Goldmann. The 3rd author gratefully acknowledges the discussions with Noga Zewi that led to the separation of tree-like and general resolution.

The 1st author was supported by the Academy of Finland (grants 132812 and 251170). The 3rd author was supported by Swedish Research Council grant 621-2010-4797 and by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no 279611. The 4th author was supported by a Junior Research Fellowship at University College, Oxford. Part of this work was done during the visits of the 1st, 2nd and 4th authors to KTH supported in part by the foundations *Johan och Jakob Söderbergs stiftelse*, *Magnus Bergvalls Stiftelse*, *Stiftelsen Längmanska kulturfonden*, and *Helge Ax:son Johnsons stiftelse*.

References

[1] Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2009)

[2] Marques-Silva, J.P., Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. IEEE Trans. Computers **48**(5) (1999) 506–521

[3] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proc. DAC, ACM (2001) 530–535

[4] Silva, J.P.M., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. [1] 131–153

[5] Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM **5**(7) (1962) 394–397

[6] Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**(3) (1960) 201–215
[7] Buss, S.R., Hoffmann, J., Johannsen, J.: Resolution trees with lemmas: Resolution refinements that characterize DLL-algorithms with clause learning. Logical Methods in Computer Science **4**(4:13) (2008)
[8] Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artificial Intelligence **175** (2011) 512–525
[9] Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. Journal of Artificial Intelligence Research **40** (2011) 353–373
[10] Alekhnovich, M., Razborov, A.A.: Resolution is not automatizable unless W[P] is tractable. In: Proc. FOCS, IEEE (2001) 210–219
[11] Bonet, M.L., Galesi, N.: Optimality of size-width tradeoffs for resolution. Computational Complexity **10**(4) (2001) 261–276
[12] Atserias, A., Dalmau, V.: A combinatorial characterization of resolution width. Journal of Computer and System Sciences **74**(3) (2008) 323–334
[13] Ben-Sasson, E., Nordström, J.: Short proofs may be spacious: An optimal separation of space and length in resolution. In: Proc. FOCS, IEEE (2008) 709–718
[14] Ansótegui, C., Bonet, M.L., Levy, J., Manyà, F.: Measuring the hardness of SAT instances. In: Proc. AAAI, AAAI Press (2008) 222–228
[15] Esteban, J.L., Torán, J.: Space bounds for resolution. Inf. Comput. **171**(1) (2001) 84–97
[16] Esteban, J.L., Torán, J.: A combinatorial characterization of treelike resolution space. Information Processing Letters **87**(6) (2003) 295–300
[17] Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: Proc. IJCAI, Morgan Kaufmann (2003) 1173–1178
[18] Nordström, J.: Pebble games, proof complexity and time-space trade-offs. LMCS (2012) to appear. Available at `http://www.csc.kth.se/~jakobn/research`.
[19] Haken, A.: The intractability of resolution. Theoret. Comp. Sci. **39**(2-3) (1985) 297–308
[20] Urquhart, A.: Hard examples for resolution. Journal of the ACM **34**(1) (1987) 209–219
[21] Ben-Sasson, E., Wigderson, A.: Short proofs are narrow—resolution made simple. Journal of the ACM **48**(2) (2001) 149–169
[22] Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Space complexity in propositional calculus. SIAM Journal on Computing **31**(4) (2002) 1184–1211
[23] Ben-Sasson, E., Galesi, N.: Space complexity of random formulae in resolution. Random Structures and Algorithms **23**(1) (2003) 92–109
[24] Nordström, J.: New wine into old wineskins: A survey of some pebbling classics with supplemental results. Foundations and Trends in Theoretical Computer Science (2012) to appear. Draft version available at `http://www.csc.kth.se/~jakobn/research/`.
[25] Ben-Sasson, E., Nordström, J.: Understanding space in proof complexity: Separations and trade-offs via substitutions. In: Proc. ICS. (2011) 401–416
[26] Nordström, J.: Narrow proofs may be spacious: Separating space and width in resolution. SIAM Journal on Computing **39**(1) (2009) 59–121
[27] Nordström, J., Håstad, J.: Towards an optimal separation of space and length in resolution (Extended abstract). In: Proc. STOC. (2008) 701–710
[28] Gilbert, J.R., Tarjan, R.E.: Variations of a pebble game on graphs. Technical Report STAN-CS-78-661, Stanford University (1978)
[29] Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. Journal of Artificial Intelligence Research **22** (2004) 319–351
[30] Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. SAT. Volume 2919 of LNCS., Springer (2004) 502–518
[31] Biere, A.: Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. FMV Tech Report 10/1, Johannes Kepler University (2010)

[32] Dilkina, B.N., Gomes, C.P., Sabharwal, A.: Backdoors in the context of learning. In: Proc. SAT. Volume 5584 of LNCS., Springer (2009) 73–79
[33] Beame, P., Beck, C., Impagliazzo, R.: Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. In: Proc. STOC, ACM (2012) 213–232