Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution

Albert Atserias[1], Johannes Klaus Fichte[2], and Marc Thurley[2]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
[2] Humboldt Universität zu Berlin, Berlin, Germany

Abstract. We offer a new understanding of some aspects of practical SAT-solvers that are based on DPLL with unit-clause propagation, clause-learning, and restarts. On the theoretical side, we do so by analyzing a concrete algorithm which we claim is faithful to what practical solvers do. In particular, before making any new decision or restart, the solver repeatedly applies the unit-resolution rule until saturation, and leaves no component to the mercy of non-determinism except for some internal randomness. We prove the perhaps surprising fact that, although the solver is not explicitely designed for it, it ends up behaving as width-k resolution after no more than n^{2k+1} conflicts and restarts, where n is the number of variables. In other words, width-k resolution can be thought as n^{2k+1} restarts of the unit-resolution rule with learning. On the experimental side, we give evidence for the claim that this theoretical result describes real world solvers. We do so by running some of the most prominent solvers on some CNF formulas that we designed to have resolution refutations of width k. It turns out that the upper bound of the theoretical result holds for these solvers and that the true performance appears to be not very far from it.

1 Introduction

The discovery of a method to introduce practically feasible clause learning to DPLL-based solvers [15, 10] layed the foundation of what is sometimes called "modern" SAT-solving. These methods set the ground for new effective implementations [11] that spawned tremendous gains in the efficiency of SAT solvers with many practical applications. Such great and somewhat unexpected success seemed to contradict the widely assumed intractability of SAT, and at the same time uncovered the need for a formal understanding of the capabilities and limitations underlying these methods.

Several different approaches have been suggested in the literature for developing a rigorous understanding. Among these we find the proof-complexity approach, which captures the power of SAT solvers in terms of propositional proof systems [3, 4, 9], and the rewriting approach, which provides a useful handle to reason about the properties of the underlying algorithms and their correctness

[1] Partially supported by project CICYT TIN2007-68005-C04-03.

[12]. In both approaches, SAT solvers are viewed as algorithms that search for proofs in some underlying proof system for propositional logic. With this view in mind, it was illuminating to understand that the proof system underlying modern solvers is always a subsystem of resolution [3]. In particular, this means that their performance can never beat resolution lower bounds, and at the same time it provides many explicit examples where SAT solvers require exponential time. Complementing this is the observation that an idealized SAT solver that relies on non-determinism to apply the techniques in the best possible way will be able to perform as good as general resolution [4, 9]. As the authors in [4] put it, the negative proof complexity results uncover examples of inherent intractability even under perfect choice strategies, while the positive proof complexity results give hope of finding a good choice strategy.

In this work we add a new perspective to this kind of rigorous result. On one hand we try to avoid non-deterministic choices on all components of our abstract solver and still get positive proof complexity results. On the other hand, we test the theoretical results experimentally with some of the available solvers, on benchmarks designed for the purpose. Our main finding is that a concrete family of SAT solvers that do not rely on non-determinism besides mild randomness is at least as powerful as bounded width resolution. The precise proof-complexity result is that under the unit-propagation rule, the totally random branching strategy, and a standard learning scheme considered by true solvers, $8k\ln(8n)n^{2k}$ conflicts and deterministic restarts are enough to detect the unsatisfiability of any CNF formula on n variables having a width-k resolution refutation, with probability at least $1/2$. Note that this bound is *not* asymptotic. The experimental results with actual solvers seem to confirm that our model is faithful enough. We discuss these at the end of this introduction.

The theoretical result by itself has some nice consequences, which we shall sketch briefly. First, it is not very surprising that, although not explicitely designed for that purpose, SAT-solvers are able to solve instances of 2-SAT in very reasonable time. The reason for this is that every unsatisfiable 2-CNF formula has a resolution refutation of width two. More strongly, our result can be interpreted as showing that width-k resolution can be simulated by $O(k\log(n)n^{2k})$ rounds of unit-clause propagation. To our knowledge, such a tight connection between width-k resolution and repeated application of "width-one" methods was unknown before. Another consequence is that SAT solvers are able to solve formulas of bounded branch-width (and hence bounded treewidth) in polynomial time. We elaborate on these later in the paper. Finally, from the partial automatizability results in [5], it follows that SAT solvers are able to solve formulas having polynomial-size tree-like resolution proofs in quasipolynomial time, and formulas having polynomial-size general resolution proofs in subexponential time.

Concerning the techniques, it is perhaps surprising that the proof of our main result does not proceed by showing that the width-k refutation is learned by the algorithm. For all we know the produced proof has much larger width. All we show is that every width-k clause in the refutation is *absorbed* by the algorithm,

which means that it behaves as if it had been learned, even though it might not. In particular, if a literal and its complement are both absorbed, the algorithm correctly declares that the formula is unsatisfiable. This analysis is the main technical contribution of this paper, and deviates significantly from [4] and [9].

Before we close this introduction, a few words on the experimental results are in order. We considered six of the most popular available SAT solvers: BerkMin 5.61 [8], MinSAT 2 [7], Siege ver. 4 [14], zChaff 2001.2.17 (32-Bit version), ZChaff 2007.3.12. (64-Bit) [11] and RSat 2.02 [13]. We ran each of these solvers with specially designed families of unsatisfiable formulas. The formulas come parameterized by two integers q and k and are designed in such a way that the number of variables n is roughly k^2q, the number of clauses is roughly $8k^2q$, and they have width-k resolution refutations. The outcome from the experiment appears to be that the number of decisions (and hence conflicts) that the solvers make on those formulas is bounded by a function of the form n^{c_k}, where c_k is a real number that is in fact smaller but comparable to k.

2 Preliminaries

A *literal* is a propositional variable x or its negation $\bar{x}$. We use the notation x^0 for $\bar{x}$ and x^1 for x. Note that x^a is defined in such a way that the *assignment* $x = a$ satisfies it. For $a \in \{0, 1\}$, we also use $\bar{a}$ for $1 - a$, and for a literal $\ell = x^a$ we use $\bar{\ell}$ for x^{1-a}. A *CNF formula* F is a set of clauses which in turn are sets of literals. The width of a clause is the number of literals in it. For two clauses $A = \{x, \ell_1, \ldots, \ell_r\}$ and $B = \{\bar{x}, \ell'_1, \ldots, \ell'_s\}$ we define the *resolvent of A and B* by $\mathrm{Res}(A, B) = \{\ell_1, \ldots, \ell_r, \ell'_1, \ldots, \ell'_s\}$. We further write $\mathrm{Res}(A, B, x)$ if we want to refer to the variable which we *resolve on*. For a clause C, a variable x, and a truth value $a \in \{0, 1\}$, the *restriction* of C on $x = a$ is the constant $\mathbf{1}$ if the literal x^a belongs to C, and the clause obtained from C by deleting any occurrence of the literal x^{1-a} otherwise. We write $C|_{x=a}$ for the restriction of C on $x = a$. A *partial assignment* is a sequence of assignments $(x_1 = a_1, \ldots, x_r = a_r)$ with all variables distinct. If S is a partial assignment and C is a clause, we let $C|_S$ be the result of applying the restrictions $x_1 = a_1, \ldots, x_r = a_r$ to C. Clearly the order does not matter. We say that S *satisfies* C if it sets at least one of its literals to 1; i.e., if $C|_S = \mathbf{1}$. We say that S *falsifies* C if it sets all its literals to 0; i.e., if $C|_S = \emptyset$. If D is a set of clauses, we let $D|_S$ denote the result of applying the restriction S to each clause in D, and removing the resulting $\mathbf{1}$'s. We call $D|_S$ the *residual* set of clauses.

3 Algorithm and Resolution Width

3.1 Definition of the algorithm

A *state* is a sequence of assignments $(x_1 = a_1, \ldots, x_r = a_r)$ in which all variables are distinct and some assignments are marked as *decisions*. We use the notation $x_i \stackrel{\mathrm{d}}{=} a_i$ to mean that the assignment $x_i = a_i$ is a *decision assignment*.

In this case x_i is called a *decision variable*. The rest of assignments are called *implied assignments*. We use the letters S and T to denote states. The empty state is the one without any assignments.

The algorithm maintains a current state S and a current database of clauses D. There are four modes of operation DEFAULT, CONFLICT, UNIT, and DECISION. Here is what the algorithm is required to do in each mode:

- DEFAULT. Check if S satisfies every clause in D, in which case stop and output SAT together with the current state S. Otherwise, check if S falsifies some clause in D, in which case move to CONFLICT mode. If not all clauses are satisfied and none of the clauses is falsified, move to UNIT mode. Finally, if control reaches this point, move to DECISION mode.
- CONFLICT. Apply the *learning scheme* to add a new clause to D. Then apply the *restart policy* to decide whether to continue further or to restart in DEFAULT mode with S initialized to the empty state and the current D. In case we continue further, find the most recently added (or conflict-causing) decision $x \stackrel{\text{d}}{=} a$ in S, if such exists. If none is found, stop and output UNSAT. If one is found, replace it by $x = \overline{a}$, delete all later assignments from S, and go back to DEFAULT mode.
- UNIT. For any clause in D for which S gives value to all its literals but one, say x^a, add $x = a$ to the current state and go back to DEFAULT mode.
- DECISION. Apply the *branching strategy* to determine a decision $x \stackrel{\text{d}}{=} a$ to be added to the current state, and go back to DEFAULT mode.

The algorithm is started in DEFAULT mode with the empty state as the current state and the given CNF formula F as the current database.

The well-known DPLL-procedure is the special case of this algorithm in which the learning scheme never adds any new clause, the restart policy does not dictate any restart at all, and the branching strategy chooses the first (or any other) variable that is still unset in the current state. Note that unit-propagation is enforced greedily before every decision is made in accordance to practical implementations. Modern SAT-solvers enhance the performance of the DPLL-procedure by implementing non-trivial learning schemes, restart policies, and branching strategies, as well as a technique known as *backjumping*. This is the mechanism by which the solver in CONFLICT mode determines which conflict-causing decision to backtrack on, based on the clause that the learning scheme adds to the database. We discuss our choice for these components of the algorithm in Section 3.3.

3.2 Runs of the algorithm

Consider a run of the algorithm started in DEFAULT mode with the empty state and initial database D, until a clause is falsified and thus a conflict occurs. Such a run is called a *round started with* D and we represent it by the sequence of states $S_0, \ldots, S_m$ that the algorithm goes through, where S_0 is the empty state and S_m is the state where the falsified clause is found. Note that for $i \in \{1, \ldots, m\}$, the

state S_i extends S_{i-1} by exactly one assignment of the form $x_i = a_i$ or $x_i \stackrel{\text{d}}{=} a_i$ depending on whether UNIT or DECISION is executed at that iteration.

A *partial round* is an initial segment $S_0, \ldots, S_r$ of a round up to a state where one of the following is true for the residual database $D|_{S_r}$: either $D|_{S_r}$ has no clauses left, or $D|_{S_r}$ contains the empty clause, or $D|_{S_r}$ does not contain any unit clause. If one of the first two cases occurs we say that the partial round is *conclusive*. If a partial round is not conclusive we call it *unconclusive*. We say that the partial round *satisfies* a clause if its last state, interpreted as a partial assignment, satisfies it. We say that it *falsifies* it if its last state, interpreted as a partial assignment, falsifies it. Note that a round may neither satisfy nor falsify a clause.

One important feature of partial rounds is that if they are unconclusive, then the residual database $D|_{S_r}$ does not contain unit clauses and, in particular, it is *closed* under unit propagation. This means that for an unconclusive partial round $S_0, \ldots, S_r$ started with D, if A is a clause in D and S_r falsifies all its literals but one, then S_r must satisfy the remaining literal, and hence A as well. Besides those in D, other clauses may have this property, which is important enough to deserve a definition:

Definition 1. *Let D be a set of clauses and let A be a non-empty clause. We say that D* absorbs *A if for every literal ℓ in A and every unconclusive partial round $S_0, \ldots, S_r$ started with D, if S_r falsifies $A \setminus \{\ell\}$, then it satisfies A.*

We argued already that every clause in D is absorbed by D. We give an example showing that D may absorb other clauses. Let D be the database consisting of the three clauses

$$a \vee \bar{b} \qquad b \vee c \qquad \bar{a} \vee \bar{b} \vee d \vee e.$$

In this example, the clause $a \vee c$ is absorbed by D but does not belong to D. Also the clause $\bar{b} \vee d \vee e$ is not absorbed by D (consider the partial round that starts by $d \stackrel{\text{d}}{=} 0, e \stackrel{\text{d}}{=} 0$) but is a consequence of D (resolve the first and the third clause on a).

The following lemma states three nice monotonicity properties of the concept of clause-absorption, where the first is the one that motivated its definition.

Lemma 1. *Let D and E be sets of clauses and let A and B be non-empty clauses. The following hold:*

1. *if A belongs to D, then D absorbs A,*
2. *if $A \subseteq B$ and D absorbs A, then D absorbs B,*
3. *if $D \subseteq E$ and D absorbs A, then E absorbs A.*

Proof. To prove 1. assume for contradiction that there is a literal ℓ in A and an unconclusive partial round $S_0, \ldots, S_r$ started with D which falsifies $A \setminus \{\ell\}$ but does not satisfy A. As the round is unconclusive, we cannot have $A|_{S_r} = \emptyset$, which means then that $A|_{S_r} = \{\ell\}$, in contradiction to the definition of partial round.

For the proof of 2. let ℓ be a literal of B and define $B' = B \setminus \{\ell\}$. We consider two different cases. If $\ell \notin A$ then $A \subseteq B'$ and, as A is absorbed by D, there is no unconclusive partial round which falsifies B'. Thus B is absorbed in this case. If $\ell \in A$, let $A' = A \setminus \{\ell\}$ and let $S_0, \ldots, S_r$ be an unconclusive partial round started with D which falsifies B'. Then it falsifies A' and satisfies A by absorption. Thus it satisfies B, and B is absorbed in this case as well.

It remains to prove 3. Let ℓ be some literal in A and $A' = A \setminus \{\ell\}$. Let $T_0, \ldots, T_r$ be an unconclusive partial round started with E which contradicts A'. We have to prove that this partial round satisfies A. To do this, define a partial round $S_0, \ldots, S_r$ started with D as follows. The first state is the empty state $S_0 = T_0$. The state S_{i+1} extends S_i by a single *extension* assignment. If the last assignment in T_{i+1} is a decision, then this will be the extension assignment. If the last assignment $x = a$ in T_{i+1} is not a decision, the unit clause $\{x^a\}$ must be in $E|_{T_i}$. Now if $\{x^a\}$ is also in $D|_{S_i}$, we let the extension assignment be $x = a$, and if not, we let it be $x \stackrel{\mathrm{d}}{=} a$.

It follows straightforwardly from this definition that $S_0, \ldots, S_r$ is an unconclusive partial round started with D. Further S_r falsifies A' as T_r does. And as A is absorbed by D it is satisfied by S_r. Therefore it is also satisfied by T_r which completes the proof. □

The following lemma describes how the resolvent of two absorbed clauses might look if it stays unabsorbed. We say that a partial round $S_0, \ldots, S_r$ *branches* in a set of literals C if all decision variables of S_r are variables of C.

Lemma 2. *Let D be a set of clauses, let A and B be two resolvable clauses that are absorbed by D, and let $C = \mathrm{Res}(A, B)$. If C is non-empty and not absorbed by D, then the following hold:*

1. *there is a literal x^a which occurs in both A and B,*
2. *there is an unconclusive partial round R started with D that falsifies $C \setminus \{x^a\}$,*
3. *the partial round R branches in $C \setminus \{x^a\}$ and leaves x unassigned, and*
4. *extending R by the decision $x \stackrel{d}{=} \bar{a}$ yields a conclusive round.*

Proof. Let $A = \{\ell, \ell_1, \ldots, \ell_p\}$ and $B = \{\bar{\ell}, \ell'_1, \ldots, \ell'_q\}$. As C is non-empty and not absorbed by D, there is a literal x^a in C and an unconclusive partial round $T_0, \ldots, T_s$ started with D which falsifies $C' = C \setminus \{x^a\}$ but does not satisfy C. In particular x is not assigned a in T_s. Also x is not assigned $\bar{a}$ in T_s since otherwise, as A and B are absorbed by D, both ℓ and $\bar{\ell}$ would be satisfied by T_s. This shows that x is unassigned in T_s.

Assume without loss of generality that x^a belongs to A, let $A' = A \setminus \{x^a\}$ and define $B' = B \setminus \{\bar{\ell}\}$. We also have that x^a belongs to B. To see this, observe that otherwise T_s would falsify B', implying that ℓ is falsified by T_s as B is absorbed by D. Then T_s falsifies A' and thus x is set to a in T_s, this time because A is absorbed by D. This contradicts the previous argument that x is unassigned in T_s. Altogether we have that x^a occurs in both A and B.

We still need to prove the existence of an unconclusive partial round $S_0, \ldots, S_r$ as stated in the Lemma. We will construct this round from the given unconclusive one $T_0, \ldots, T_s$. As T_s falsifies C' and x is unassigned in T_s the sole issue we

need to resolve is the possibility that T_s might not branch in C'. We will define $S_0, \ldots, S_r$ inductively. It will be convenient to also define an offset j_i for each $i \in \{0, \ldots, r\}$. Recall that S_0 is the empty state by definition. We define $j_0 = 0$. To construct S_{i+1}, let $h > j_i$ be the minimum number in $\{0, \ldots, s\}$ such that the h-th assignment in T_s is of one of the following types, if it exists:

1. a decision $y \stackrel{\mathrm{d}}{=} b$ for some variable y from C',
2. an implied assignment $y = b$, and $\{y^b\}$ is a unit clause in $D|_{S_i}$,
3. an implied assignment $y = b$, and y is a variable from C'.

If no such h exists, we stop the construction and let $r = i$. If such an h exists, we define $j_{i+1} = h$ and S_{i+1} from S_i by cases. In the first of the three cases above, let S_{i+1} be obtained from S_i by adding the decision $y \stackrel{\mathrm{d}}{=} b$. In the second case, the assignment is due to the existence of the unit clause $\{y^b\}$ in $D|_{T_{h-1}}$. As $\{y^b\}$ is also a unit clause in $D|_{S_i}$, we define S_{i+1} as the extension of S_i by adding this assignment. If the first two cases do not occur, we must be in the third, and we define S_{i+1} from S_i by extending it with $y \stackrel{\mathrm{d}}{=} b$.

Clearly, this defines a valid partial round $S_0, \ldots, S_r$ which branches in C'. Further S_r falsifies C' and all the assignments in S_r also appear in T_s except for some additional "decision" marks on some assignments. Therefore the partial round is unconclusive and the variable x is unassigned in S_r. Finally, as the assignments S_r and T_s are the same with respect to the literals in A and B, extending the partial round $S_0, \ldots, S_r$ by a decision $x \stackrel{\mathrm{d}}{=} \bar{a}$ yields a conclusive round; otherwise both ℓ and $\bar{\ell}$ would be satisfied since both A and B are absorbed by D. □

The following lemma will allow us to turn the existential statement about a partial round R in Lemma 2 into a universal statement about all partial rounds R' that make decisions that are already in R. If R and R' are partial rounds, we say that the decisions of R' are subsumed by R if every decision assignment in R' is also an assignment in R. We say that R' is subsumed by R if every assignment made in R' is also an assignment in R.

Lemma 3. *Let D be a set of clauses, let R be an unconclusive partial round started with D, and let R' be a partial round started with D with all its decisions subsumed by R. Then R' is subsumed by R.*

Proof. Let $S_0, \ldots, S_r$ and $T_0, \ldots, T_s$ be the partial rounds R and R', respectively. Assume for contradiction that R' is not subsumed by R. Then there exists a minimal $i \in \{1, \ldots, s\}$ such that all assignments made in T_{i-1} are also in made in R, but the last assignment in T_i is not made in R. Since every decision assignment in R' is also an assignment in R, the last assignment in T_i must be an implied one of the form $x = a$. Thus, there exists a unit clause $\{x^a\}$ in $D|_{T_{i-1}}$. As every assignment in T_{i-1} is also made in R, and as R is unconclusive and does not contain $x = a$ or $x \stackrel{\mathrm{d}}{=} a$, there exists a $j \in \{0, \ldots, r\}$ such that this unit clause is also present in $D|_{S_j}$. Finally, since R is an unconclusive partial round, $D|_{S_r}$ does not contain unit clauses and thus $x = a$ is also an assignment in S_r. Contradiction. □

One consequence of this lemma is that under the hypothesis and the notation of Lemma 2, if x^a and R are the literal and the unconclusive partial round claimed to exist in that lemma, then every partial round started with D that has all its decisions subsumed by R stays unconclusive, and if in addition it ends up falsifying $C \setminus \{x^a\}$, then it yields a conclusive round after extending it with $x \stackrel{\text{d}}{=} \bar{a}$. Indeed, the extension $x \stackrel{\text{d}}{=} \bar{a}$ would force the round to satify both ℓ and $\bar{\ell}$ as both A and B are absorbed by hypothesis. We will need this fact in what follows.

3.3 Restart policy, learning scheme, and branching strategy

The only really important issue of the *restart policy* that we want to use is that it should dictate restarts often enough. As a matter of fact, we will state and prove our result for the most aggressive of all restart policies, the one that dictates a restart at every conflict, and the analysis will extend to other restart policies by monotonicity. More precisely, by the monotonicity properties discussed in the previous section, it will follow from our analysis that if we decide to use a policy that allows $c > 1$ conflicts per round before a restart, then the upper bound on the number of required restarts can only decrease (or stay the same). Only the upper bound on the number of conflicts would appear multiplied by a factor of c, even though the truth might be that even those decrease as well. One further consequence of monotonicity is that the validity of our analysis is insensitive to whether the solver implements *backjumping* or not. For the same reason as before, allowing $c > 1$ conflicts per round with their corresponding backjumps can only decrease the number of required restarts in our analysis, and multiply the number of conflicts by a factor of c. Thus, for the rest of the paper, we fix the restart policy to the one that restarts at every conflict.

Let us discuss now the *learning scheme*. This determines which clause to add to the database in the CONFLICT mode of the algorithm. We will consider the scheme called DECISION in the literature, that obtains the clause by the following method. Let $S_0, \ldots, S_m$ be a conclusive round started with the clause database D that ends up falsifying some clause of D. We anotate each state S_i of the round by a clause A_i by reverse induction on $i \in \{1, \ldots, m\}$:

1. For $i = m$, let A_i be the first clause in D that is falsified by S_i.
2. For $i < m$ for which $x_i \stackrel{\text{d}}{=} a_i$ is a decision, let $A_i = A_{i+1}$.
3. For $i < m$ for which $x_i = a_i$ is implied, let B_i be the first clause in D which contains literal $x_i^{a_i}$ and for which S_{i-1} gives value to all its literals but one, and let $A_i = \text{Res}(A_{i+1}, B_i, x_i)$ if these clauses are resolvable on x_i, and let $A_i = A_{i+1}$ otherwise.

It is quite clear from the construction that each A_i has a resolution proof from the clauses in the database D. In fact, the resolution proof is linear and even trivial in the sense of [4]. The learning scheme called DECISION is the one that adds the clause A_1 to the current database after each conflict. It is not hard to check that every literal in A_1 is the negation of some decision literal in S_m; this will be important later on.

The *branching strategy* determines which literal x^a is branched next in the DECISION mode of the algorithm. We will analyse the totally random branching strategy defined as follows: if the current state of the algorithm is S, we choose a variable x uniformly at random among the variables that appear in the residual database $D|_S$, and a value a in $\{0,1\}$ also uniformly at random and independently of x. Our analysis actually applies to any other branching strategy that randomly chooses between making a heuristic-based decision or a random decision as above, provided the second case has non-negligible probability of happening. If $p \in (0,1]$ is the probability of the second case, the bounds in our analysis will appear multiplied by a factor of p^{-k}, where k is the resolution width that we are trying to achieve.

3.4 Resolution width

We start by analysing the number of rounds it takes until the resolvent of two absorbed clauses is absorbed as a function of its width.

Lemma 4. *Let D be a database of clauses, and let A and B be two resolvable clauses that are absorbed by D and that have a non-empty resolvent $C = \mathrm{Res}(A,B)$. Then, for every integer $t \geq 0$, the probability that C is not absorbed by the database after t restarts is at most $e^{-t/4n^k}$, where n is the total number of variables in D and k is the width of C.*

Proof. Let $D_0, D_1, \ldots, D_t$ be the sequence of databases produced by the algorithm, starting with $D = D_0$. By the monotonicity properties in Lemma 1, if C is ever absorbed by some D_i it will stay so until D_t. Thus, it will suffice to bound the probability that D_{i+1} does not absorb C conditional on the event that D_i does not absorb C.

Assume D_i does not absorb C. By Lemma 2, there exists a literal x^a in $A \cap B \cap C$ and an unconclusive partial round R started with D_i that falsifies $C \setminus \{x^a\}$, branches in $C \setminus \{x^a\}$, leaves x unassigned, and the extension of R by $x \stackrel{\mathrm{d}}{=} \bar{a}$ yields a conclusive round. Moreover, by Lemma 3 and the discussion after it, any partial round R' that has all its decisions subsumed by R stays subsumed by R and unconclusive, and if it ends up falsifying $C \setminus \{x^a\}$, then its extension by $x \stackrel{\mathrm{d}}{=} \bar{a}$ will also yield a conclusive round. Such a round would yield a conflict that makes the DECISION scheme learn a subclause of C, which implies that C would be absorbed by D_{i+1} by Lemma 1.

First, let us compute a lower bound on the probability that the first $k-1$ choices of the branching strategy falsify $C \setminus \{x^a\}$ and that the k-th choice is $x \stackrel{\mathrm{d}}{=} \bar{a}$. This probability is at least

$$\left[\left(\frac{k-1}{2n}\right)\left(\frac{k-2}{2(n-1)}\right)\cdots\left(\frac{1}{2(n-k+2)}\right)\right]\left(\frac{1}{2(n-k+1)}\right) \geq \frac{1}{4n^k}.$$

Note that a round following these choices may not even be able to do some of the decisions as the corresponding assignments may be implied. However, before

the decision $x \stackrel{\mathrm{d}}{=} \bar{a}$, the round will only perform decisions that are subsumed by R and therefore stay subsumed by R by Lemma 3. In particular it will stay unconclusive and x will remain unset. It follows that the probability that the round will start by branching in, and falsifying, $C \setminus \{x^a\}$, and end by deciding $x \stackrel{\mathrm{d}}{=} \bar{a}$ can only increase. This gives a lower bound on the probability that a subclause of C is actually learned, and with it, the probability that C is not absorbed by D_{i+1} is bounded by $1 - \frac{1}{4n^k}$.

By chaining these t conditional probabilities, the probability that C is not absorbed by D_t is bounded by

$$\left(1 - \frac{1}{4n^k}\right)^t \leq e^{-t/4n^k},$$

as was to be proved. □

Finally, we are ready to state and prove the main result of the paper.

Theorem 1 *Let F be a set of clauses on n variables having a resolution refutation of width k and length m. With probability at least $1/2$, the algorithm started with F learns the empty clause after at most $4m\ln(4m)n^k$ conflicts and restarts.*

Proof. The resolution refutation must terminate with an application of the resolution rule of the form $\mathrm{Res}(x, \bar{x})$. We will show that for both $\ell = x$ and $\ell = \bar{x}$, the probability that $\{\ell\}$ is not absorbed by the current database after $4m\ln(4m)n^k$ restarts is at most $1/4$. Thus, both $\{x\}$ and $\{\bar{x}\}$ will be absorbed with probability at least $1/2$. If this is the case, it is straightforward that every round of the algorithm is conclusive. In particular, the round that does not make any decision is conclusive, and in such a case the empty clause is learned.

Let $C_1, C_2, \ldots, C_r = \{\ell\}$ be the resolution proof of $\{\ell\}$ that is included in the width-k resolution refutation of F. In particular $r \leq m - 1$ and every C_i is non-empty and has width at most k. Let $D_0, D_1, \ldots, D_s$ be the sequence of databases produced by the algorithm where $s = rt$ and $t = \lceil 4\ln(4r)n^k \rceil$. For every $i \in \{0, \ldots, r\}$, let E_i be the event that every clause in the initial segment $C_1, \ldots, C_i$ is absorbed by D_{it}, and let $\overline{E}_i$ be its negation. Note that $\Pr[\, E_0 \,] = 1$ vacuously and hence $\Pr[\, \overline{E}_0 \,] = 0$. For $i > 0$, we bound the probability that E_i does not hold conditional on E_{i-1} by cases. Let $p_i = \Pr[\, \overline{E}_i \mid E_{i-1} \,]$ be this probability. If C_i is a clause in F, we have $p_i = 0$ by Lemma 1. If C_i is derived from two previous clauses, we have $p_i \leq e^{-t/4n^k}$ by Lemma 4, which is at most $1/4r$ by the choice of t.

The law of total probability gives

$$\begin{aligned}\Pr\left[\, \overline{E}_i \,\right] &= \Pr\left[\, \overline{E}_i \mid E_{i-1} \,\right] \Pr\left[E_{i-1}\right] + \Pr\left[\, \overline{E}_i \mid \overline{E}_{i-1} \,\right] \Pr\left[\, \overline{E}_{i-1} \,\right] \\ &\leq \Pr\left[\, \overline{E}_i \mid E_{i-1} \,\right] + \Pr\left[\, \overline{E}_{i-1} \,\right].\end{aligned}$$

Adding up over all $i \in \{1, \ldots, r\}$, together with $\Pr\left[\overline{E}_0\right] = 0$, gives

$$\Pr\left[\, \overline{E}_r \,\right] \leq \sum_{i=1}^{r} p_i \leq \frac{r}{4r} = \frac{1}{4}.$$

Since the probability that C_r is not absorbed by D_{rt} is bounded by $\Pr[\ \overline{E}_r\ ]$, the proof follows. □

The total number of clauses of width k on n variables is bounded by $2^k\binom{n}{k}$, which is at most $2n^k$ for every n and k. Therefore, if F has n variables and a width-k resolution refutation, we may assume that its length is at most $2n^k$. We obtain the following consequence:

Corollary 1. *Let F be a set of clauses on n variables having a resolution refutation of width k. With probability at least $1/2$, the algorithm started with F learns the empty clause after at most $8k\ln(8n)n^{2k}$ conflicts and restarts.*

An application of Corollary 1 is that, even though it is not explicitely defined for the purpose, the algorithm can be used to decide the satisfiability of CNF formulas of treewidth at most k in time $O(k\log(n)n^{2k+2})$. This follows from the known fact that every unsatisfiable formula of treewidth at most k has a resolution refutation of width at most $k+1$ [1, 6, 2]. If we are interested in producing a satisfying assignment when it exists, we proceed by self-reducibility: we assign variables one at a time, running the algorithm $\log_2(n)+1$ times at each iteration to detect if the current partial assignment cannot be extended any further, in which case we choose the complementary value for the variable. For this we use the fact that if F has treewidth at most k, then $F|_{x=a}$ also has treewidth at most k. Note that each iteration is correct with probability at least $1-1/2n$, which means that all iterations are correct with probability at least $1/2$. The running time of this algorithm is $O(k(\log(n))^2n^{2k+3})$.

4 Experiments on Tseitin Formulas

In this section we will discuss the experiments performed to illustrate our theoretical results. The class of formulas we tested are Tseitin formulas on *trees of k-grids*. To give a precise definition let the *k-grid* be a graph $G_k=(V_k,E_k)$ with vertex set $V_k=\{v_{i,j}\mid i,j\in[k]\}$ and edges $E_k=\{\{v_{i,j},v_{i',j'}\}\mid |i-j|-|i'-j'|=1\}$. Let further $k'=\lfloor k/2\rfloor$ and define $\{v_{1,1},\ldots,v_{1,k'}\}$ as the set of *top* vertices and $\{v_{k,1},\ldots,v_{k,k'}\}$ and $\{v_{k,k-k'},\ldots,v_{k,k}\}$ that of *left* and *right* bottom vertices of G_k. In a given rooted binary tree $\mathcal{T}$ we associate with each node t a distinct labelled k-grid G_t. Then if t has a child t' the top vertices of $G_{t'}$ are merged with the left bottom vertices of G_t by identifying $v'_{1,i}$ with $v_{k,i}$ for all $i\in[k']$. A second child t'' is treated analogously by now merging the right bottom vertices of G_t with the top vertices of $G_{t''}$.

For any tree of k-grids $G=(V,E)$ as defined above, we construct an unsatisfiable Tseitin CNF-formula F_G. The construction is well-known an can be found e.g. in [16]. Note that the number of variables of F_G is roughly $n=k^2|V|$. Further, the resolution width of F_G is at most k.

Randomized formulas. To average running times of SAT solvers on the above formulas, we introduce some randomization. Let $q\in\mathbb{N}$. A *random* binary tree $\mathcal{T}$ contains a root r and is constructed as follows. Then for every node t assume

that we know the number $q' > 0$ of its descendants to be constructed. Choose $q'' \leq q'$ u.a.r. and recursively construct two subtrees, one with q'' nodes, the other one with $q' - q''$ nodes. The process stops if $q' = 0$. For $q, k \in \mathbb{N}$ a random Tseitin formula $F_{q,k}$ is a formula F_G for some tree of k-grids G which in turn has been constructed from a random binary tree on q nodes.

4.1 Results

We conducted experiments using several SAT solvers on a Linux machine with a 3.0 GHz Pentium 4 processor and 1 GB of RAM. The solvers tested include BerkMin 5.61 [8], MinSAT 2 [7], Siege ver. 4 [14], zChaff 2001.2.17 (32-Bit version), zChaff 2007.3.12. (64-Bit) [11] and RSat 2.02 [13]. As running times of the solvers increase quickly with the parameter k, we chose to consider different test sets for the different solvers.

Small values $k = 2, \ldots, 5$. For each k we generated instances $F_{q,k}$ with q varying from 1 to 101 in steps of 10 with 100 instances per step. Note that for $k = 5$ and $q = 100$ a formula $F_{q,k}$ already contains about 4000 variables and 14000 clauses.

Average running times for solving instances $F_{q,k}$ with $k = 5$ and $q = 100$ are as small as 20s for RSat, whereas MiniSat timed out after 10000s. We are however not interested in the actual running times of the solver but we aim at quantifying the difficulty (as a function of k) of solving these formulas.

We therefore chose to consider the average number of decisions with respect to the number n of variables of the formulas $F_{q,k}$. Under the hypothesis that for fixed k the number of decisions d is bounded by a polynomial in n, we determined, for each solver and each n the minimum $c = c(k)$ such that $d \leq n^c$. The experimental results show that this c is a function $c = c(k, n)$ of k and n. The dependence of c on n is significant especially for small formulas. However, for fixed k and large n it turns out that the value of $c(k, n)$ is quite stable. For example, on formulas with more than 100 tree nodes we observed that the oscillation of $c(k, n)$ did never exceed 10%. Figure 2 displays the exponents for RSat, which, for comparability, are given in terms of the tree nodes q.

Altogether, it turns out that for fixed k the number of decisions of the solvers is bounded by a polynomial $n^{c(k)}$. Figure 1 illustrates these values of $c(k)$ for the different solvers. The actual values were determined for $q = 100$, which we chose as some solvers turned out to have problems solving much larger instances.

In particular, for $k = 5$ MiniSAT was not able to solve instances with $q \geq 60$ within 10000s therefore the corresponding value has been excluded from Figure 1. Further, although the 2001 zChaff solved many instances for $k = 4, 5$ and arbitrary q within the given time bounds, most of these instances could not be solved due to out-of-memory errors.

Larger values $k = 6, \ldots, 8$. The running times of most solvers quickly exceeded 10000s. Therefore we generated very sparse sets of test instances, mainly to show the tendency of the growth of the running time.

For $k = 6$ we generated a test set with $q = 1, \ldots 101$ in steps of 10 with 10 instances per step. On these formulas BerkMin showed a peculiar behaviour in so

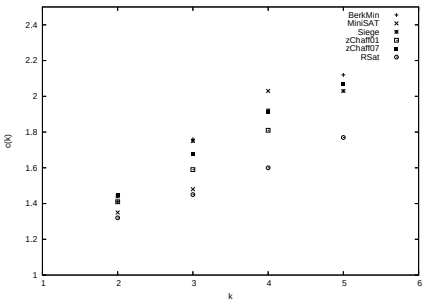

Fig. 1. The value $c = c(k)$.

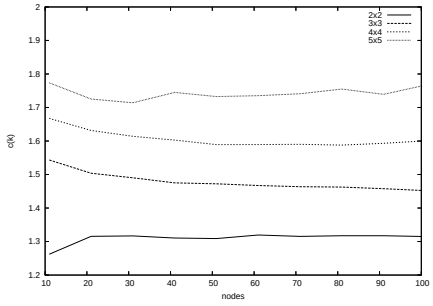

Fig. 2. Stabilization of the Degree

grid sz.	BerkMin	MiniSAT	Siege	zChaff01	zChaff07	RSat2.0
2×2	1.41	1.35	1.44	1.41	1.45	1.32
3×3	1.76	1.48	1.75	1.59	1.68	1.45
4×4	1.92	2.03	1.92	1.81^*	1.91	1.60
5×5	2.12	2.62^*	2.03	$-^*$	2.07	1.77
6×6	–	–	2.22	–	2.55^*	1.91
7×7	–	–	2.39	–	–	2.04^*
8×8	–	–	2.63^*	–	–	2.13^*

Table 1. The value $c = c(k)$. For the marks * see the discussion in the text.

far as it was able to solve most instances of up to 6 tree-nodes within 50s although starting at 7 nodes it was not at all able to solve any instance within 10000s. The 2007 version of zChaff was much more stable, but the average running time exceeded 10000s at 30 tree nodes. The exponent in the table was taken for $q = 51$ where the average running time exceeded even 31000s.

The test set for $k = 7$ was identical to that for $k = 6$. Only Siege and RSat remained for testing. Siege was able to finish the test set. For RSat, the exponent was determined at $q = 81$, since at $q = 91$ out of memory errors occurred. For $k = 8$ we generated a test set of $1, \ldots 51$ tree nodes in steps of 10 and 5 instances per step. The average running time of both Siege and RSat was about 60000s at 51 tree nodes.

Discussion. The test set for smaller k seems to confirm the theoretical results of the previous section. The growth of the decisions of all solvers is polynomial for each fixed k and the exponent of this running time grows at most linearly with k. For Siege and RSat this growth even seems to be mildly sublinear, although exact analysis of this fact would necessitate more detailed tests. Note that the number of decisions is always at least that of the conflicts. Thus the number of conflicts is bounded as predicted by Theorem 1.

However, it is not possible to draw a safe conclusion from these results. Especially by the sparsity of the test set for large k, we cannot take the results to be more than an illustration of the link we assume between true SAT solvers and our theoretical results.

5 Future Work

Our theoretical results establish a correlation between restarts and width, and the experimental results indicate that real-world solvers seem tuned in a way that exploits this correlation. The experiments are however at an early stage and further work will be necessary before drawing definitive conclusions. First, one should try larger test sets and larger values of the parameter k. A second particularly urgent matter is that our experiments do not count restarts directly; they count conflicts, which is only an upper bound on the number of restarts. Related to this is the question of testing the different solvers with different restart policies to compare their behaviour with the theoretical prediction. This is perhaps the most promising open end for applications of our theoretical investigation. Third, an important pressing issue is the lack of a truly general model of randomized formulas of a given width. This is, indeed, a question of theoretical interest by itself.

Acknowledgements

The authors would like to thank Martin Grohe for two reasons. First for giving the idea for the class of formulas used in the experimental part. Most importantly we thank him for the conjecture which became the main result of this paper.

References

1. Michael Alekhnovich and Alexander A. Razborov. Satisfiability, branch-width and tseitin tautologies. In *FOCS*, pages 593–603. IEEE Computer Society, 2002.
2. Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *J. Comput. Syst. Sci.*, 74(3):323–334, 2008.
3. Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Understanding the power of clause learning. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 1194–1201. Morgan Kaufmann, 2003.
4. Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22:319–351, 2004.
5. Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. In *STOC*, pages 517–526, 1999.
6. Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 310–326, London, UK, 2002. Springer-Verlag.

7. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.
8. E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat-solver. *Design, Automation and Test in Europe (DATE'02)*, 2002.
9. Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively p-simulate general propositional resolution. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 283–290. AAAI Press, 2008.
10. Robert J. Bayardo Jr. and Robert C. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the Fourtheenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.
11. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.
12. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
13. Knot Pipatsrisawat and Adnan Darwiche. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.
14. Lawrence Ryan. Efficient algorithms for clause-learning sat solvers. Master's thesis, Simon Fraser University, 2004.
15. Joao P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.
16. Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.