Electronic Colloquium on Computational Complexity, Report No. 108 (2013)

New Algorithms for QBF Satisfiability and Implications for Circuit Complexity

Rahul Santhanam
University of Edinburgh

Ryan Williams
Stanford University

Abstract

We revisit the complexity of the satisfiability problem for quantified Boolean formulas. We show that satisfiability of quantified CNFs of size poly(n) on n variables with $O(1)$ quantifier blocks can be solved in time $2^{n-n^{\Omega(1)}}$ by zero-error randomized algorithms. This is the first known improvement over brute force search in the general case, even for quantified polynomial-sized CNFs with one alternation. The algorithm gives an improvement over 2^n when the number of quantifier blocks is $q = o(\log\log n/\log\log\log n)$. We also show how to achieve non-trivial savings on formulae when the number of quantifier blocks is $q = \omega(\log n)$.

Next, we study the time complexities of QBF satisfiability over CNF formulas versus QBF over arbitrary Boolean formulas. We present surprisingly strong relationships between these time complexities, showing how to efficiently express Boolean formulas as quantified CNFs. As a consequence, the two problems have essentially equivalent time complexities in many cases, and further improvements over brute force search for quantified CNF satisfiability would imply breakthroughs in circuit complexity. For example, if satisfiability of quantified CNF formulae with n variables, poly(n) size and at most q quantifier blocks can be solved in time $2^{n-n^{\omega_q(1/q)}}$, then NEXP does not have $O(\log n)$ depth circuits of polynomial size. Furthermore, solving satisfiability of quantified CNF formulae with n variables, poly(n) size and $O(\log n)$ quantifier blocks in time $2^{n-\omega(\log(n))}$ time would imply the same circuit complexity lower bound. Therefore, substantial improvements on the algorithms of this paper would imply new circuit complexity lower bounds.

ISSN 1433-8092

1 Introduction

The satisfiability (SAT) problem for Boolean formulas is the canonical NP-complete problem. Despite its apparent worst-case intractability, nowadays the ability to solve SAT instances in practice is critical for many practical applications. Over the past two decades, there have been many substantial advances in SAT solvers that led to the current state of affairs that "SAT is generally easy in practice" [MZ09]. On the theoretical side, there are many known SAT algorithms which provably solve the problem faster than brute force search for formulas in conjunctive normal form (CNF) [MS85, PPZ97, Pud98, PPSZ98, Sch99, Sch05, DW06, CIP09, IMP12].

The quantified Boolean formula (QBF) problem is the analogue of SAT for the larger complexity class PSPACE. Variables can have arbitrary quantification (existential or universal), and the problem is to determine whether a given quantified formula is true or false. QBF would potentially have a much wider range of applications than SAT, if only we understood more about how to solve it. The best known QBF solvers can only tackle a very limited range of the problem space [Zha06, GIB09]. Moreover, in theory, comparatively very little is known concerning general worst-case algorithms for QBF [Wil02, San10, CIP10]. Although improved algorithms were known for quantified CNFs with cn clauses and n variables (for constant c), slightly more general problems remained wide open. Even for CNF formulas with n variables and $\text{poly}(n)$ clauses, it was open whether QBF was solvable in faster than 2^n time on formulas of the form

$$(\forall x_1)\cdots(\forall x_k)(\exists x_{k+1})\cdots(\exists x_n)\phi$$

where ϕ is 3-CNF. (Such formulas are said to have *two quantifier blocks*, or *one quantifier alternation*.) Calabro, Impagliazzo, and Paturi [CIP10] gave evidence that this special case will be very hard to solve: they showed that general CNF satisfiability on n variables can be reduced in subexponential time to quantified kCNF on $n + O(n^{1/(k-1)})$ variables with two quantifier blocks. Therefore, any 1.9^n time algorithms for quantified 3CNF would imply similar results for CNF-SAT, resolving a longstanding open question.[1]

In this paper, we present new generic algorithms for solving QBF on CNF and DNF formulas that run faster than brute force, as well as interesting hardness reductions demonstrating that these problems are surprisingly powerful. Our algorithms exploit several observations from the literature along with some new methods of analysis. Our hardness results show that quantified Boolean CNFs are a highly expressive class of logical formulas compared to the usual CNFs, giving a partial explanation for why QBFs in practice are so much more difficult to solve.

Beating Brute Force for QBF Satisfiability. We first show that for n-variable CNF with significantly less than $\log n$ quantifier alternations, there are algorithms which can beat brute force search.

Theorem 1.1 *Satisfiability of quantified CNF (or DNF) formulas with q quantifier blocks, n variables, and $m \leq 2^{n^{1/(q+1)^{q-1}}}$ clauses (or disjuncts) can be solved probabilistically with zero error in time $O(\text{poly}(m) \cdot 2^{n-\Omega(n^{\delta_q}/(\log m))})$, where $\delta_q = 1/(q+1)^{q-1}$.*

It follows that brute-force search can be beaten for all QBFs with up to $o(\log\log n/\log\log\log n)$ quantifier alternations. We conjecture that δ_q in the theorem can in fact be improved to $1/q$, which would beat brute force even for $q = o(\log n/\log\log n)$. For the case of $q = 2$, we can achieve this for 3-CNFs:

[1] The Strong Exponential Time Hypothesis posits that CNF-SAT does not have a 1.9^n time algorithm [IP01, CIP06, DW10]. Nevertheless, it is also known that even minor improvements over exhaustive search, such as $O(2^n/n^4)$, can lead to circuit lower bounds in certain cases [Wil10, Wil11, Wil13].

Theorem 1.2 *Satisfiability of quantified 3-CNF (or 3-DNF) formulas with two quantifier blocks, n variables, and m clauses (or disjuncts) can be solved deterministically in time $O(\text{poly}(m) \cdot 2^{n-\Omega(\sqrt{n})})$.*

This is an interesting complement to the hardness reduction of Calabro, Impagliazzo, and Paturi [CIP09], who reduce arbitrary CNF SAT on n variables and $O(n)$ clauses to an instance of the above problem with only $n + O(\sqrt{n})$ variables.

Next, we show that for formulas with asymptotically *more* than $\log n$ quantifier alternations, a simple randomized algorithm beats 2^n:

Theorem 1.3 *Let $r : \mathbb{N} \to \mathbb{N}$ satisfy $r(n) < n$ for all n. Satisfiability of $r(n)$-QBF over $\text{poly}(n)$-size circuits with n input variables can be solved probabilistically with zero-error in $O(\text{poly}(n)2^{n-\Omega(r(n))})$ time.*

Theorems 1.1 and 1.3 show how to beat brute force on quantified CNFs with up to $O\left(\frac{\log\log n}{\log\log\log n}\right)$ quantifier alternations, and those with $\omega(\log n)$ quantifier alternations. Furthermore, our conjecture that Theorem 1.1 can be improved would handle the case of $O(\log n/\log\log n)$ alternations, but not $O(\log n)$. Is this merely a weakness in our ways of reasoning about QBF, or is the case of $O(\log n)$ quantifiers an especially difficult one?

The Log-Alternation Barrier. Surprisingly, we show that improving over exhaustive search for the log-alternation case is indeed difficult! We first show that beating brute force on n-variable QBF instances with $O(\log n)$ alternations, even in CNF form, would imply faster algorithms for satisfiability of arbitrary polynomial-size Boolean formulas over arbitrary gates of fan-in two. More precisely, we show how to efficiently express poly-size Boolean formulas as QBFs in CNF with $O(\log n)$ quantifiers and $n + O(\log n)$ variables.

Theorem 1.4 *There is a polynomial time algorithm that takes any Boolean formula of n inputs and s size and outputs an equivalent quantified CNF instance of $n + O(\log s)$ variables, $O(s^4)$ clauses, and $O(\log n)$ quantifier blocks (alternations).*

Combining this with a reduction of [CIP09], we obtain:

Corollary 1.1 *There is a polynomial time algorithm that, for every fixed k, takes any Boolean formula of n inputs and s size and outputs an equivalent quantified kCNF instance of $n + O(\log s) + O(s^{4/(k-1)})$ variables, $\text{poly}(s)$ clauses, and $O(\log n)$ quantifier blocks (alternations).*

Applying work of Williams [Wil11], it follows that faster solution of quantified kCNFs with $O(\log n)$ alternations would imply that NEXP does not have $\text{poly}(n)$-size $O(\log n)$-depth circuits, a major open problem in circuit complexity.

Corollary 1.2 *If for all k, quantified CNF with n variables, n^k clauses, and $k \log n$ alternations can be solved in zero-error probabilistic $2^n/n^k$ time, then $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\text{poly}$.*

Corollary 1.3 *If there is an $\varepsilon > 0$ such that for all k, quantified kCNF with n variables, n^k clauses, and $k \log n$ alternations can be solved in zero-error probabilistic 2^{n-n^ε} time, then $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\text{poly}$.*

That is, such algorithms imply that NEXP does not have $O(\log n)$ depth circuits of polynomial size. For the constant-quantifier case, it is natural to ask if the $n^{\Omega(1)}$ savings in the exponent Theorem 1.1 can be improved upon, without proving new lower bounds. Again, we can give a negative answer by showing

that even quantified CNFs with a *constant* number of alternations can be very expressive. Extending Theorem 1.4, we show that every formula of polynomial size can be logically expressed as a quantified CNF with $n + O(n^{1/k})$ variables and only $O(k)$ quantifier alternations.

Theorem 1.5 *Let $k, r > 0$ be any integers. There is a polynomial-time algorithm that takes any Boolean formula of n inputs and depth $r\log(n)$, and outputs an equivalent quantified CNF instance of $n + O(n^{1/k})$ variables, size $\mathrm{poly}(n)$, and $2kr$ quantifier blocks.*

Since any formula of size s can be expressed as a formula of depth $O(\log s)$, the parameter r in the theorem can be made $O(1)$, yielding $O(k)$ quantifier blocks. As a consequence:

Corollary 1.4 *If quantified CNF with n variables, $\mathrm{poly}(n)$ clauses, and q alternations can be solved in zero-error probabilistic time $2^{n-n^{\omega_q(1/q)}}$, then $\mathsf{NEXP} \not\subseteq \mathsf{NC}^1/\mathrm{poly}$.*

We conclude that any substantial improvement over our algorithms would imply new lower bounds in circuit complexity. We do not wish to suggest that such algorithms do not exist, but rather that they will have very interesting consequences for complexity theory.

2 Preliminaries

Given a function $s : \mathbb{N} \to \mathbb{N}$, let $\mathsf{SIZE}(s)$ denote the class of Boolean functions with bounded fan-in circuits of size $O(s)$, and $FormulaSIZE(s)$ denote the class of Boolean functions with fan-in 2 formulas of size $O(s)$ over the De Morgan basis. Given a depth function $d : \mathbb{N} \to \mathbb{N}$, let $\mathsf{DEPTH}(d)$ denote the class of Boolean functions with fan-in 2 Boolean circuits of depth d over the De Morgan basis.

Lemma 2.1 ([Spi71, BB94]) *Let s be a size function such that $n \leq s(n)$ for all n. Then $\mathsf{FormulaSIZE}(s) \subseteq \mathsf{DEPTH}(2.1\log(s))$.*

As we will deal with several different quantified formula problems, it is important to establish notation distinguishing between them. We define QB-kCNF, QB-CNF, QB-FORMULAS to be the quantified Boolean formula problems over k-CNF predicates, arbitrary CNF predicates, and formula predicates, respectively. The case of a bounded number of quantifier blocks is often studied in complexity theory and logic:

Definition 2.1 *A quantified Boolean formula ϕ has q* quantifier blocks *or $q-1$* alternations *if it has the form*

$$\phi = (Q_1\ x_1, \ldots, x_{t_1})(Q_2\ x_{t_1+1}, \ldots, x_{t_1+t_2}) \cdots (Q_k\ x_{t_1+\cdots+t_{q-1}+1}, \ldots, x_{t_1+\cdots+t_q})F,$$

where each $Q_i \in \{\exists, \forall\}$.

The problems q-QB-kCNF, q-QB-CNF, and q-QB-DNF denote the restriction of the QBF problem over these predicates to formulas with at most q quantifier blocks.

In satisfiability problems, the choice of logical predicate can play a major role in the time complexity of solving the problem: for every k there is a $\delta < 1$ such that k-SAT is in $2^{\delta n}$ time (e.g., [MS85]), but no 1.999^n time algorithm is known for general CNF-SAT. Indeed the Strong Exponential Time Hypothesis [IP01, CIP09] posits that none exists. For satisfiability problems over more expressive predicates such as arbitrary formulas, there is no algorithm known to run in time faster than $O(2^n)$ for polynomial-size formulas. In fact if Formula SAT were in $O(2^n/n^{10})$ time, then NEXP would not be in $\mathsf{NC}^1/\mathrm{poly}$ [Wil11]).

In contrast, our hardness results show that quantified k-CNFs are essentially *equivalent* in time complexity to quantifiers over arbitrary formulas. This is a very different picture from the complexity of SAT.

3 Algorithms

3.1 Quantified Formulas with a Bounded Number of Quantifier Blocks

Our algorithm for q-QB-CNF (quantified CNFs with a bounded number of quantifier blocks q) builds on two existing algorithms from the literature. The first is a known algorithm for CNF satisfiability.

Theorem 3.1 ([DW06], building on [Sch05]) *There is a deterministic algorithm A solving satisfiability of CNFs with m clauses and n variables in time $O(\text{poly}(m)2^{n-n/(1+\log(m))})$.*

This is not the best known time bound for CNF SAT (slightly better bounds are proved in [DH09, CIP09]) but it will suffice for our algorithm. The second algorithm solves satisfiability of AC^0 circuits, i.e., constant-depth circuits over unbounded fan-in AND and OR gates, with negations on the variables at the bottom.

Theorem 3.2 ([IMP12]) *There is a probabilistic zero-error algorithm B solving satisfiability of constant-depth AND/OR circuits of n variables, s size, and d depth, in time $O(s \cdot 2^{n-\Omega(n/(\log s)^{d-1})})$.*

We begin with an algorithm for quantified CNFs and DNFs with q quantifier blocks.

Reminder of Theorem 1.1 *Satisfiability of q-QB-CNF (resp. q-QB-DNF) with n variables and $m \leq 2^{n^{1/(q+1)^{q-1}}}$ clauses (resp. disjuncts) can be solved probabilistically in time $\text{poly}(m) \cdot 2^{n-\Omega(n^{1/(q+1)^{q-1}}/(\log m))}$ with zero error.*

Proof. First, we observe that the most difficult cases of q-QB-CNF occur when the innermost quantifier block is existential; otherwise, it is easy to tell whether universally quantified variables over a CNF evaluate to true or false (such a QBF is true if and only if the CNF is a tautology over the universal variables, i.e., the empty formula with no clauses). Similarly, the difficult case of q-QB-DNF occurs when the innermost quantifier is universal.

Therefore, without loss of generality, we may assume the input QBF ϕ has the form

$$\exists \vec{x_1} \forall \vec{x_2} \ldots \vec{x_q} \psi(\vec{x_1} \ldots \vec{x_q}),$$

where ψ is a DNF with m clauses if q even, a CNF with m clauses if q odd. (If this is not the case, we can negate the formula and enforce it.) The number of variables n equals $\sum_{i=1}^{q} |\vec{x_i}|$. We assume q is even for simplicity (so $\vec{x_q}$ is universally quantified); the case of odd q is analogous. For each i such that $1 \leq i \leq q$, let $n_i = |\vec{x_i}|$.

If $n_q \geq n^{1/(q+1)^{q-1}}/q$, then the algorithm cycles over all binary strings of length $\Sigma_{j=1}^{j=q-1} n_i$, substitutes the corresponding variable assignment for $\vec{x_1} \ldots \vec{x_{q-1}}$ into ψ to obtain a simplified formula ψ' which is over the variable set $\vec{x_q}$, and runs the Algorithm A for CNF-SAT from Theorem 3.1 on the formula $\neg\psi'$, which can be converted to a CNF of $O(m)$ clauses using De Morgan's rules. The algorithm constructs an AND-OR tree of assignments to all variable blocks excluding $\vec{x_j}$, with the results of invocation of Algorithm A at the leaves, and then evaluates the AND-OR tree exhaustively to obtain the answer for ϕ. Each invocation of Algorithm A returns in time $\text{poly}(m) \cdot 2^{n_q - n_q/(1+\log(m))}$. There are $2^{\Sigma_{i=1}^{q-1} n_i}$ invocations, which yields a running time of $\text{poly}(m) \cdot 2^{n-\Omega(n^{1/(q+1)^{q-1}}/\log(m))}$.

Otherwise, $n_q < n^{1/(q+1)^{q-1}}/q$. Let $1 \leq i < q$ be the largest integer such that $n_i \geq n^{1/(q+1)^{i-1}}/q$ and $n_{i+1} < n^{1/(q+1)^i}/q$. Such an i exists, by the bound on n_q, together with the fact that $\Sigma_{j=1}^{n} n_j = n$. The algorithm constructs an AC^0 circuit C of depth $q - i + 2$ which has as input $\vec{x_1} \ldots \vec{x_i}$.

- If i is odd, C is an AND over all possible assignments to $\vec{x_{i+1}}$, of an OR over all possible assignments to $\vec{x_{i+2}}$, ..., of an AND over all possible assignments to $\vec{x_q}$ of the DNF predicate $\psi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_q})$.

- If i is even, C is an OR over all possible assignments to $\vec{x_{i+1}}$, of an AND over all possible assignments to $\vec{x_{i+2}}$, ..., of an AND over all possible assignments to $\vec{x_q}$ of the DNF predicate $\psi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_q})$.

The circuit C has depth $q - i + 2$ and size at most $2^{\sum_{j=i+1}^{q} n_j} \cdot m$.

The algorithm cycles over all possible assignments to $\vec{x_1}, \vec{x_2} \ldots \vec{x}_{i-1}$, and for such assignment, it runs Algorithm B for AC^0 SAT on C with input $\vec{x_i}$ if i is odd, and on $\neg C$ with input $\vec{x_i}$ if i is even, flipping the output of Algorithm B in the latter case. It constructs an AND-OR tree of assignments to all variable blocks preceding $\vec{x_i}$, with the results of invocation of Algorithm B at the leaves, and then evaluates the AND-OR tree exhaustively to obtain the answer for ϕ. Each invocation of Algorithm B returns in time at most $O(m2^{\sum_{j=i+1}^{q} n_j}) \cdot 2^{n_i - \Omega(n_i/(\sum_{j=i+1}^{q} n_j + \log m)^{(q-i+1)})}$. Since $\sum_{j=i+1}^{q} n_j = O(n^{1/(q+1)^i})$ and $\log m \leq n^{1/(q+1)^{q-1}}$, the time bound is at most $O(2^{\sum_{j=i+1}^{q} n_j}) \cdot 2^{n_i - \Omega(n_i/(n^{1/(q+1)^i})^{(q-i+1)})}$. Finally, because $n_i \geq n^{1/(q+1)^{i-1}}/q$ and $i \geq 1$, the runtime bound for one invocation of B is at most $O(2^{\sum_{j=i}^{q} n_j - \Omega(n^{1/(q+1)^i})})$. As the number of invocations of Algorithm B is $2^{\sum_{j=1}^{i-1} n_j}$, thus the running time of the algorithm is at most $O(2^{n - \Omega(n^{1/(q+1)^{q-1}})})$. □

Note that Theorem 1.1 gives a non-trivial improvement over brute force even when the number of quantifier blocks is $q = (\log\log n)/(\log\log\log n)$. For QBF over 3-CNF with two quantifier blocks, the above algorithm only yields $2^{n - \Omega(n^{1/3}/\log m)}$ time. With a more sophisticated algorithm, we can improve the $n^{1/3}/\log n$ factor to $n^{1/2}$.

Reminder of Theorem 1.2 *Satisfiability of 2-QB-3CNF (or 2-QB-3DNF) with n variables and* poly(n) *clauses can be solved deterministically in time $2^{n-\Omega(n^{1/2})}$.*

Proof. We will run a recursive algorithm R on the given 3CNF formula with n variables and two quantifier blocks.

In the following, we let n denote the number of variables in the *original* formula fed to the algorithm R. (Although the number of variables in the current formula may decrease over multiple calls to R, the parameter n stays the same throughout the algorithm.) Let $\epsilon > 0$ be a constant which we fix later.

0. If there is a clause containing only universal literals or an empty clause, return 0. If there are no clauses left to satisfy, return 1. If there are no universal literals in the formula, the instance is a 3SAT formula. If the 3SAT is satisfiable then return 1 else return 0.

1(a). Let u denote the number of universal variables and e be the number of existential variables. If $e > \sqrt{n}$, then try all 2^{u-e} assignments to the universals and solve the remaining 3CNF in 1.4^e time, using (for example) the deterministic 3-SAT algorithm of Moser and Scheder [MS11]. This case takes $O^\star(2^u 1.4^{-e}) = 2^{n - \Omega(\sqrt{n})}$ time.

1(b). *[At this point, the number of existential variables is at most $\sqrt{n}$.]* Suppose there is a clause of length three with two universal literals and one existential. Let it be $(u_i \vee u_j \vee e_k)$, where u_i, u_j are universal and e_k is existential. Perform the three recursive calls:

- Set $(u_i = 1)$ and call the algorithm R on the formula,
- Set $(u_i = 0)$ and $(u_j = 1)$ and call R,
- Set $(u_i = 0)$, $(u_j = 0)$, $(e_k = 1)$ and call R.

If all three calls return 1 then return 1, else return 0.

1(c) *[At this point, all clauses contain at most one universal literal.]* If $u < \epsilon n$, then solve the QBF using brute force search. Otherwise, make a DNF with 2^e conjuncts, consisting of an OR over all variable assignments to the e remaining existential variables, and each conjunct is an AND over the remaining universal literals in each clause, given a fixed existential variable assignment. This DNF has $O(2^e \cdot u)$ clauses, and u variables. The tautology problem for this DNF can be solved in $O^\star(2^{u-\Omega(u/\log(2^e))}) \leq O^\star(2^{u-\Omega(u/e)})$ time using the CNF-SAT algorithm of Theorem 3.1. Return 1 if the DNF is a tautology, else return 0.

Let $T_e(u)$ be the running time of the algorithm R on a formula with n variables initially, and u universal variables and e existential variables. Every run of the algorithm R on a given formula can be expressed as follows. Either $e > \sqrt{n}$ at the beginning of the algorithm, in which case the algorithm R runs in $2^{n-\Omega(\sqrt{n})}$ time and halts, or $e \leq \sqrt{n}$ and we have a recursion tree representing the recursive calls in case 1(b). (Note that if $e \leq \sqrt{n}$ in the very first call to R, then we also have $e \leq \sqrt{n}$ in every subformula, so only cases 1(b) and 1(c) will be applied after that.) At the leaves of the recursion tree (when there is at most one universal literal in each clause), a procedure of $2^{u-\Omega(u/e)}$ time is applied; we associate each leaf with a *cost*, namely the corresponding running time at that leaf. The *running time* for the recursion tree is bounded by the sum of the costs over all leaves of the recursion tree.

We term the three recursive calls in case 1(b) calls of type A, type B and type C respectively. We first bound the size N, i.e., the number of leaves, of the recursion tree. We classify leaves according to how many calls of type C occur on the path from root to leaf. For each i such that $0 \leq i \leq \sqrt{n}$, let $f(i)$ be the number of leaves for which i calls of type C occur on the path from root to leaf. Note that at most $\sqrt{n}$ calls of type C occur on any path from root to leaf, as $e \leq \sqrt{n}$ at the root and e decreases by 1 for each call of type C.

Observe that $f(0) = O(\phi^u) = O(\phi^n)$, where ϕ is the golden ratio, since if no calls of type C occur, there are only calls of type A and type B, the first of which decreases u by 1 and the second of which decreases u by 2. Thus the recurrence for $f(0)$ as a function of u is the same as the Fibonnaci recurrence.

In general, we have that $f(i) \leq \binom{n}{i} O(\phi^u)$, as there are $\binom{n}{i}$ ways of choosing the i levels of the recursion tree at which calls of type C are made, and among these, the size of the recursion tree is largest, namely $O(\phi^u)$, when all calls of type C are made before calls of type A and type B. Thus we have that

$$N \leq \sum_{i=0}^{i=\sqrt{n}} f(i) \leq 2^{o(n)} \phi^u \leq 2^{n-\Omega(n)},$$

since $\binom{n}{i} = 2^{o(n)}$ for $i \leq \sqrt{n}$, and $\phi^u \leq \phi^n \leq 2^{n-\Omega(n)}$. Let $\delta > 0$ be a constant such that $N \leq 2^{n-\delta n}$.

We now partition the set of leaves into *deep* leaves and *shallow* leaves, defined as follows. Let $\gamma = \delta/2$. Deep leaves are those which occur at depth at least $(1-\gamma)n$, and shallow leaves are leaves which are not deep. We will account for the number of deep leaves and shallow leaves separately.

First we analyze the deep leaves. Consider any deep leaf at depth $d \geq (1-\gamma)n$. The cost of any deep leaf is at most $2^{\gamma n}$. Since there are at most $2^{n-\delta n}$ total leaves in the recursion tree, there are at most $2^{n-\delta n}$ deep leaves, and the total cost of all such leaves is at most $2^{n-\delta n+\gamma n} = 2^{n-\Omega(n)}$.

Now set $\epsilon = \delta/4$. We partition the shallow leaves into two classes – *heavy* and *light* leaves: heavy leaves are shallow leaves with $u > \epsilon n$, and light leaves are shallow leaves which are not heavy.

The cost of any light leaf is at most $2^{\epsilon n}$. Because light leaves are shallow, there are at most $2^{(1-\gamma)n}$ light leaves, and hence the cumulative cost of such leaves is at most $2^{n-\gamma n+\epsilon n} = 2^{n-\delta n/4} = 2^{n-\Omega(n)}$.

Finally, we consider the heavy leaves. The cost of a heavy leaf at depth d is at most

$$2^{u-\Omega(u/e)} \leq 2^{n-d-\Omega(\sqrt{n})},$$

since for such leaves we have $u \leq n - d$, $u \geq \epsilon n$ and $e \leq \sqrt{n}$. Thus the cumulative cost of all heavy leaves is at most $2^{n-\Omega(\sqrt{n})}$.

Summing up all contributions, the total cost of all leaves of the recursion tree is at most $O(2^{n-\Omega(\sqrt{n})})$, establishing the running time of the algorithm. □

Based on the above analysis, we conjecture that the algorithm can be extended to q quantifiers for all q:

Conjecture 3.1 *For all constant k, satisfiability of q-QB-3-CNF with n variables and* $\text{poly}(n)$ *clauses can be solved deterministically in time $2^{n-\Omega(n^{1/q})}$.*

3.2 Faster Algorithm for Quantified Formulas with Many Quantifiers

Theorem 1.1 gives improvements over brute force search for QBF satisfiability on formulae with a small number of quantifiers. Somewhat counterintuitively, we observe that it is also possible to get savings when the number of quantifiers is relatively large, by a simple application of the idea of Snir [Sni85] and Saks-Wigderson [SW86] for reducing the decision tree complexity of game-tree evaluation.

Reminder of Theorem 1.3 *Let $q : \mathbb{N} \to \mathbb{N}$ satisfy $q(n) < n$ for all n. Satisfiability of $q(n)$-QBF over m-size circuits with n input variables can be solved probabilistically in* $\text{poly}(m) \cdot 2^{n-\Omega(q(n))}$ *time, with zero-error.*

Proof Sketch of Theorem 1.3. The idea is to explore the tree of variable assignments randomly. In particular, we branch on each variable in turn according to the quantifier order of the QBF, picking a random 0-1 assignment to the variable and recursing on the resulting instance. If the current variable is quantified existentially and the call returns SAT, then we return SAT, else we flip the variable to its opposite value and recurse, returning the answer resulting from the call. Dually, if the current variable is universally quantified and a recursive call returns UNSAT, then we return UNSAT, else we flip the variable and recurse on it.

We can see that random search saves $\Omega(1)$ per alternation on average in the exponent of the running time, using two simple observations. First, if the QBF instance is SAT, then at most $1/2$ fraction of the possible assignments to variables in each existential quantifier block are explored on average, before finding a certificate that the instance is satisfiable (i.e., a recursive call that returns SAT). Second, if the QBF is UNSAT, then at most $1/2$ fraction of the assignments in each universal quantifier block are explored on average, before finding a certificate that the instance is unsatisfiable (i.e., a call that returns UNSAT). No matter whether the given QBF instance is SAT or UNSAT, there is a $\Omega(1)$ savings in the exponent of the running time for every two consecutive quantifier blocks. These savings yield $\Omega(q(n))$ savings in the exponent when there are $q(n)$ quantifer blocks. □

Note that the above algorithm only improves over exhaustive search when $r(n) = \omega(\log(n))$: otherwise, the $\text{poly}(n)$ factor dominates the the $2^{-\Omega(r(n))}$ savings. This is an interesting phenomenon where the presence of many quantifiers creates so many additional constraints on the QBF problem that the problem actually becomes easier to solve.

4 Encoding Boolean Formulas with QB-CNFs

In the second part of the paper, we investigate the consequences of finding faster QBF satisfiability algorithms. We find that Boolean formulas can be encoded rather well as QBF satisfiability instances over CNF, even bounded-width CNFs. This is extremely different from the case of typical satisfiability over 3-CNF (i.e., 3-SAT), as there are many known faster-than-2^n time algorithms for 3-SAT, but no general reductions from formula SAT to 3-SAT which yield faster SAT algorithms for general Boolean formulae. Relating these encoding results to known connections between SAT algorithms and circuit lower bounds,

we prove that faster algorithms for QBF satisfiability over CNF would imply new lower bounds in circuit complexity such as NEXP is not contained in non-uniform NC^1.

4.1 Quantified Boolean Formulae with Arbitrary Number of Quantifier Blocks

We establish a rather tight relationship between the time complexity of solving quantified formulas and that of solving quantified 3-CNF [2]. We will show that the choice of predicate in a quantified problem essentially does not matter: if quantified 3-CNF is solvable in 1.9^n time then all quantified Boolean formulas are solvable in about 1.9^n time.

Reminder of Theorem 1.4 *There is a polynomial time algorithm that takes any Boolean formula of n inputs and s size and outputs an equivalent quantified CNF instance of $n + O(\log s)$ variables, $O(s^4)$ clauses, and $O(\log n)$ quantifier blocks (alternations).*

Proof. We first do some general massaging of a given formula F, construed as a tree with interior nodes labeled by AND/OR gates, and leaves labelled by literals. We can assume WLOG that F has depth $c \log s$ where $c < 4$. (If this is not true, we can make it the case, at a cost of squaring the formula size, via a standard reduction.) Moreover, we may assume that the depth is even, and odd depths of F contain no AND gates (only OR gates, along with possibly $0 - 1$ constants and literals), while even depths contain no OR gates. (Enforcing this only increases the depth by a factor of two, and at most squares the size.) Finally, we can make the length of every path in F from the output gate to a literal *exactly* $d = 4 \log s$. (Suppose a path ends early at a literal ℓ; since $b \wedge b = b$ and $b \vee b = b$ for $b \in \{0, 1\}$, we can duplicate occurrences of ℓ to match the desired alternations of ANDs and ORs and the desired length of each path.) Notice that F now has exactly 2^d leaves.

Let d be the depth of F, and let $L = \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ be the set of literals of F. Define a mapping $\phi_F : \{0, 1\}^d \to L$ as follows. Given a d-bit string b, label every edge in F to a left child with a 0, and every edge to a right child with 1. Follow the path from the output gate (the root of the tree) by reading the bits of b from left to right, then output the literal at the leaf found. Note that by our above reductions on F, the map ϕ_F is a bijection.

The variables of our QB-CNF instance will be $x_1, \ldots, x_n$ along with new variables $y_1, \ldots, y_d$. For convenience, we use the notation $y_i^1 := y_i$ and $y_i^0 := \neg y_i$. The instance of QB-CNF includes all 2^d possible clauses of the form

$$(y_1^{1-b_1} \vee \cdots \vee y_d^{1-b_d} \vee \phi_F(b_1, \ldots, b_d)),$$

over all possible vectors $(b_1, \ldots, b_d) \in \{0, 1\}^d$. Noticing that

$$(y_1^{1-b_1} \vee \cdots \vee y_d^{1-b_d} \vee \phi_F(b_1, \ldots, b_d)) \equiv ((y_1^{b_1} \wedge \cdots \wedge y_d^{b_d}) \to \phi_F(b_1, \ldots, b_d)),$$

we have that (a) for every assignment to $(y_1, \ldots, y_d)$, at most one of the above clauses is not trivially satisfied, and (b) this remaining clause is satisfied iff the literal at the leaf defined by the root-to-leaf path $b_1 \cdots b_d$ is true. The final formula is then

$$(Q_1 x_1, \ldots, Q_n x_n)(\forall y_1)(\exists y_2) \cdots (\forall y_d)[C]$$

where C is the above collection of clauses and $Q_i \in \{\exists, \forall\}$ is the quantifier on variable x_i from the original quantified Boolean formula. □

The following two reductions to QB-CNF are immediate corollaries:

[2] The results of this subsection appeared in Section 3.3 of our ECCC technical report TR12-059. A modified and enhanced version of TR12-059, but without the results of Section 3.3, appeared in CCC 2012. We include these results here because they fit better with the theme of our paper, namely solving satisfiability for QBFs.

Corollary 4.1 *There is a polynomial time reduction from* QB-FORMULA *instances of n inputs and s size to* QB-CNF *instances of $n + O(\log s)$ variables and $O(s^4)$ size.*

Corollary 4.2 *There is a polynomial time reduction from* FORMULA-SAT *instances of n inputs and s size to* QB-CNF *instances of $n + O(\log s)$ variables, $O(s^4)$ size, and $O(\log n)$ quantifier blocks.*

These demonstrate (1) the QBF problem over arbitrary formulas and the problem over CNFs are essentially identical, and (2) formula satisfiability can be efficiently reduced to QBF over CNF formulas with only $O(\log n)$ quantifier blocks. Applying results of Williams [Wil11], it follows that a nontrivial algorithm for QB-CNF satisfiability (even with zero-error) for $O(\log n)$ quantifier blocks would imply new circuit lower bounds:

Reminder of Corollary 1.2 *If for all k, quantified CNF with n variables, n^k clauses, and $k \log n$ alternations can be solved in zero-error probabilistic $2^n/n^k$ time, then* $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\mathrm{poly}$.

Proof. Williams [Wil11] shows that if there is a co-nondeterministic algorithm for FORMULA-SAT on instances with n inputs and n^k size running in $O(2^n/n^{10})$ time for all k, then $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\mathrm{poly}$. Hence the proof follows from Corollary 4.2. □

We can reduce from quantified CNF formulas to quantified k-CNF formulas, by applying a reduction of Calabro, Impagliazzo, and Paturi [CIP10].

Theorem 4.1 ([CIP10]) *For all k, there is a polynomial-time reduction from* CNF-SAT *with n variables and m clauses to* 2-QB-kCNF *with $n + O(m^{1/(k-1)})$ variables,* $\mathrm{poly}(m, n)$ *clauses.*

Proof. (Sketch) They use $O(m^{1/(k-1)})$ additional variables to encode CNF evaluation in such a way that the given CNF F on n variables evaluates to 1 on an assignment iff the new QBF with $n+O(m^{1/(k-1)})$ variables is unsatisfiable. In particular, they define a $(k-1)$-CNF formula G with m clauses and $O(m^{1/(k-1)})$ variables which is *minimally unsatisfiable* in the sense that if any clause is removed then the formula is satisfiable. To build the clauses of the QBF, they iterate over all clauses c_i of the given F and literal ℓ in it, putting in the QBF the clause $(c_i \vee \neg\ell)$. Our final QBF existentially quantifies over the n variables of F, then univerally quantifies over the $O(m^{1/(k-1)})$ variables of G. Then, a given assignment A to the n variables of F is satisfying if and only if every clause in G is present in the QBF after variable assignment A (which is true if and only if the remaining clause set is *unsatisfiable*). □

The above proof easily extends to a reduction from QB-CNF with q quantifier blocks to QB-kCNF with $q+1$ quantifier blocks, with the same increase in the number of variables. Combining Corollary 4.1 and Theorem 4.1, we find a close relation between the time complexity of QB-kCNF and QB-FORMULA:

Reminder of Corollary 1.3 *If there is an $\varepsilon > 0$ such that for all k, quantified kCNF with n variables, n^k clauses, and $k \log n$ alternations can be solved in zero-error probabilistic 2^{n-n^ε} time, then* $\mathsf{NEXP} \not\subset \mathsf{NC}^1/\mathrm{poly}$.

4.2 Quantified Boolean Formulae with Constant Number of Quantifer Blocks

Reminder of Theorem 1.5 *Let $k, r > 0$ be any integers. There is a polynomial-time algorithm that takes any Boolean formula of n inputs and depth $r\log(n)$, and outputs an equivalent QB-CNF instance of $n + O(n^{1/k})$ variables, size* $\mathrm{poly}(n)$*, and $2kr$ quantifier blocks.*

Proof. We use a similar divide-and-conquer approach to the proof of Nepomnjascii's theorem [Nep70] that NC is in alternating linear time. Let F be the given formula of depth $r\log(n)$ on variables $x_1 \ldots x_n$. We assume without loss of generality that all leaves (i.e., literals or constants) of F are at depth $r\log(n)$; if there is a literal or constant v at lower depth d, we replace it by a full binary tree of ANDs of depth $(r\log(n) - d)$ with v at each leaf.

We imagine F as divided into kr layers, where layer i consists of all nodes between depth $\lfloor (i\log(n))/k \rfloor$ and depth $\lfloor (i\log(n)+\log(n))/k \rfloor$. Sub-formulas at layer i are those sub-trees of the formula which have as root a node at depth $\lfloor (i\log(n))/k \rfloor$ and as leaves the descendants of the root at depth $\lfloor (i\log(n)+\log(n))/k \rfloor$. We call such a sub-formula for any $i, 0 \leq i \leq kr-1$, a *local* sub-formula. Every local sub-formula has at most $4n^{1/k}$ nodes, since F is a formula where all nodes have fan-in at most two.

We construct a QB-CNF instance ϕ as follows. ϕ begins with $2kr-1$ quantifier blocks $Q_1, Q_2 \ldots Q_{2kr}$, where Q_i is existential if i is odd, and universal if i is even. The quantifier blocks Q_i for odd i quantify over variable blocks $y^1, y^2 \ldots y^{kr}$, where for each $j, 1 \leq j \leq kr$, each variable block y^j consists of $\lceil 4n^{1/k} \rceil$ variables $y^j_l, l = 1 \ldots \lceil 4n^{1/k} \rceil$. The quantifier blocks Q_i for even i quantify over variable blocks $z^1, z^2 \ldots z^{kr-1}$, where for each $j, 1 \leq j \leq kr-1$, each variable block z^j consists of $m(n) = \lceil \log(2n^{1/k}) \rceil$ variables $z^j_l, l = 1 \ldots \lceil \log(2n^{1/k}) \rceil$. Note that the total number of quantified variables is $O(n^{1/k})$. Following the quantifier blocks is a CNF formula ψ in variables $x_1 \ldots x_n, y^1 \ldots y^{kr}, z^1 \ldots z^{kr-1}$. The variables $x_1 \ldots x_n$ will be called x-variables, $y^1 \ldots y^{kr}$ will be y-variables, and $z^1 \ldots z^{kr}$ will be z-variables.

Before describing ψ, we give some intuition for the construction of the QBF instance. The idea is to partition the formula into several local sub-formulae, each of small (i.e., $O(n^{1/k})$) size, and to verify the computation of the formula by verifying each local sub-formula independently. To verify the computation of a local sub-formula, values for all the nodes in the sub-formula are guessed existentially and checked for local consistency. Of course, to guess values for all nodes of the formula independently would require too many extra variables. Instead, universal quantifiers are used to take advantage of the layered structure of the formula and re-use variables between different sub-formula verifications. It turns out that a constant number of alternations suffices for this purpose.

We define a way to inductively index local sub-formulas. We identify each local sub-formula with its root. The root of the formula F is indexed by $()$. Now, given an index $(t_1, t_2 \ldots t_i)$ for a local sub-formula T at layer i, let $T_1, T_2 \ldots T_\ell$ be the local sub-formulas at layer $i+1$ which are leaves of T, where $t_1 \ldots t_i \in \mathbb{N}$, and $\ell \leq 2n^{1/k}$. We say that $(t_1, t_2 \ldots t_i, t_{i+1})$ indexes T_j for $t_{i+1} \in \mathbb{N}$, $1 \leq j \leq \ell$, if $t_{i+1} \equiv j \mod \ell$.

Now, let $t_1, t_2 \ldots t_{kr-1} \in \{0,1\}^{m(n)}$ be arbitrary. For each such sequence, and for each initial segment $t_1, t_2 \ldots t_i$, $0 \leq i \leq kr-1$ of the sequence, we define a CNF formula $\psi_{(t_1, t_2 \ldots t_i)}$ of size poly(n). The CNF formula ψ is the conjunction of $\psi_{(t_1, t_2 \ldots t_i)}$ over all sequences and all $i, 0 \leq i \leq kr-1$. It is not hard to see that ψ is of size poly(n).

Fix i. Interpreting $t_1, t_2 \ldots t_{kr-1}$ as non-negative integers in the natural way, let T be the local sub-formula indexed by $(t_1, t_2 \ldots t_i)$. We define a CNF $cons_T$ which has as variables $y^1, y^2 \ldots y^{i+1}$ and $x_1 \ldots x_n$. We then define $\psi_{(t_1, t_2 \ldots t_i)}$ using $cons_T$ and additional variables $z^1, s^2 \ldots z^i$. In fact, $cons_T$ will only involve y-variables in y^i and y^{i+1}, and will only involve x-variables when $i = kr-1$. $\psi_{(t_1, t_2 \ldots t_i)}$ will only involve additional z-variables in z^i.

We distinguish two cases: $i < kr-1$ and $i = kr-1$. When $i < kr-1$, we define $cons_T$ as follows. Identify the variables in y^{i+1} with distinct nodes of T in some canonical way. There are at least as many variables in y^{i+1} as nodes in T, so some variables might be left over - these just won't be used in $cons_T$. Let a, b, c be arbitrary nodes in T such that b and c are children of a. Assume wlog that y^{i+1}_1, y^{i+1}_2 and Y^{i+1}_3 are the y-variables identified with a,b and c respectively. If a is an AND gate, $cons_T$ contains a set of clauses encoding that $y^{i+1}_2 \wedge y^{i+1}_3 = y^{i+1}_1$, else a is an OR-gate and $cons_T$ contains a set of clauses encoding that $y^{i+1}_2 \vee y^{i+1}_3 = y^{i+1}_1$. In addition, suppose v is the root of T, y^{i+1}_s is associated with v in T and y^i_q is associated with v in the local sub-formula of which v is a leaf. Then there is a pair of clauses in $cons_T$ encoding that $y^{i+1}_s = y^i_q$.

When $i = kr-1$, $cons_T$ contains all clauses as before, but in addition "leaf clauses" as follows. In this case, the leaves of T are all literals, i.e., x-variables or their complements, or else constants. Leaf

clauses encode, for each leaf v' of T, that the y-variable associated with v' is equal to the corresponding literal/constant in formula F.

The CNF $cons_T$ encodes local consistency of the local sub-formula T in terms of the guessed information represented by the y-variables, but since we would like to re-use the y-variables for all local sub-formulae at a given layer, we would like this consistency check to kick in only for a specific setting of the universal variables. This is ensured by incorporating the z-variables as follows.

For each $j, 1 \leq j \leq m(n)$, let $w_j = z^i_j$ if the j'th bit of t_i is 0, and let w_j be the complement of z^i_j otherwise. When $i \geq 1$, we define $\psi_{t_1 \ldots t_i}$ to be $\vee_{j=1}^{m(n)} w_j \vee cons_T$. When $i = 0$, $\psi_{t_1 \ldots t_i}$ is the same as $cons_T$.

This completes the description of the QB-CNF formula ϕ. It is clear that ϕ can be constructed from F in polynomial time. What remains to be shown is that ϕ is equivalent to F.

Suppose F evaluates to 1 on a specific input $x_1, x_2 \ldots x_n$. Then by assigning y-variables to the values of corresponding interior nodes in the formula for this assignment, it is easy to see that ϕ is satisfied. Conversely, if ϕ is satisfied, then using the fact that local consistency as well as consistency between a local sub-formula and its parent is encoded into ϕ, F evaluates to 1. Indeed, values of interior nodes for F can be recovered from any witness tree for ϕ. □

Corollary 4.3 *There is a polynomial-time reduction from* FORMULA-SAT *instances of n inputs and size* $\mathrm{poly}(n)$ *to* QB-CNF *instances with $O(k)$ quantifier blocks, $n + O(n^{1/k})$ variables and size* $\mathrm{poly}(n)$.

Corollary 4.3 follows from Theorem 1.5 using Lemma 2.1, which gives a simulation of $\mathrm{poly}(n)$ size formulas by $O(\log(n))$ depth formulas. Note that any $O(\log(n))$ depth circuit can be simulated by an $O(\log(n))$ depth formula by creating separate copies of sub-formulae to replace gates with fan-out greater than 1.

Reminder of Corollary 1.4 *If satisfiability of quantified CNF with n variables,* $\mathrm{poly}(n)$ *clauses and q quantifier blocks can be solved in zero-error probabilistic time $2^{n-n^{\omega_q(1/q)}}$, then* $\mathsf{NEXP} \not\subseteq \mathsf{NC}^1/\mathrm{poly}$.

Proof. Suppose the assumption holds. Then, using Corollary 4.3, we have that Formula-SAT instances of n inputs and size $\mathrm{poly}(n)$ can be solved in zero-error probabilistic time $2^{n-\omega(\log(n))}$. Using the algorithms-to-lower-bounds connection of Williams [Wil11], this implies that $\mathsf{NEXP} \not\subseteq \mathsf{NC}^1/\mathrm{poly}$. □

By combining Corollary 4.3 with Theorem 4.1, we can derive an even stronger connection:

Corollary 4.4 *If satisfiability of quantified q-CNF with n variables,* $\mathrm{poly}(n)$ *clauses and q quantifier blocks can be solved in zero-error probabilistic time $2^{n-n^{\omega_q(1/q)}}$, then* $\mathsf{NEXP} \not\subseteq \mathsf{NC}^1/\mathrm{poly}$.

5 Conclusions

We have given several new methods for solving quantified Boolean formulas faster than exhaustive search methods, and shown that solving the case of quantified CNF would have surprising consequences: essentially any improvement over 2^n time, even for CNFs with $O(\log n)$ quantifier blocks, would imply superpolynomial size lower bounds for log-depth circuits.

The primary open question is to improve our algorithms further. There are several new directions to explore with our approach. We conjecture that the bound of Theorem 1.1 can be improved to $2^{n-\Omega(n^{1/q})}$, and have proven the conjecture for the case $q = 2$.

A secondary but still important question is to better understand how algorithms for quantified Boolean formulas can imply circuit complexity lower bounds. In this paper we have shown that even weak improvements over exhaustive search for quantified CNF would imply NEXP does not have $O(\log n)$ depth circuits of polynomial size, but it seems likely that stronger connections can be found.

References

[BB94] Maria Bonet and Samuel Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994.

[CIP06] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of IEEE Conference on Computational Complexity*, pages 252–260, 2006.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Proc. International Workshop on Parameterized and Exact Computation*, 2009.

[CIP10] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified *k*-cnf. In *Parameterized and Exact Computation (IPEC)*, pages 50–59, 2010.

[DH09] Evgeny Dantsin and Edward Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, chapter 12, pages 403–424. IOS Press, 2009.

[DW06] Evgeny Dantsin and Alexander Wolpert. A faster clause-shortening algorithm for sat with no restriction on clause length. *JSAT*, 1(1):49–60, 2006.

[DW10] Evgeny Dantsin and Alexander Wolpert. On moderately exponential time for SAT. In *Proceedings of 13th International Conference on Satisfiability Testing*, pages 313–325, 2010.

[GIB09] Alexandra Goultiaeva, Vicki Iverson, and Fahiem Bacchus. Beyond CNF: A circuit-based QBF solver. In *Theory and Applications of Satisfiability Testing (SAT 2009)*, pages 412–426, 2009.

[IMP12] Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC0. In *Proceedings of 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 961–972, 2012.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[MS85] Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[MS11] Robin A. Moser and Dominik Scheder. A full derandomization of Schöning's k-sat algorithm. In *STOC*, pages 245–252, 2011.

[MZ09] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76–82, 2009.

[Nep70] V. Nepomnjascii. Rudimentary predicates and turing calculations. *Soviet Mathematics - Doklady*, 11(6):1462–1465, 1970.

[PPSZ98] Ramamohan Paturi, Pavel Pudlak, Mike Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. In *Proceedings of 39th International Symposium on Foundations of Computer Sciece (FOCS)*, pages 628–637, 1998.

[PPZ97] Ramamohan Paturi, Pavel Pudlak, and Francis Zane. Satisfiability coding lemma. In *Proceedings of 38th International Symposium on Foundations of Computer Science (FOCS)*, pages 566–574, 1997.

[Pud98] Pavel Pudlak. Satisfiability – algorithms and logic. In *Mathematical Foundations of Computer Science, Springer LNCS Volume 1450*, pages 129–141, 1998.

[San10] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 183–192, 2010.

[Sch99] Uwe Schoning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *Proceedings of 40th Annual Symposium on Foundations of Computer Science*, pages 410–414, 1999.

[Sch05] Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

[Sni85] Marc Snir. Lower bounds on probabilistic decision trees. *Theoretical Computer Science*, 38:69–82, 1985.

[Spi71] Philip Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the Fourth Hawaii International Symposium on System Sciences*, pages 525–527, 1971.

[SW86] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 29–38, 1986.

[Wil02] Ryan Williams. Algorithms for quantified boolean formulas. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 299–307, 2002.

[Wil10] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing*, pages 231–240, 2010.

[Wil11] Ryan Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of 26th Annual IEEE Conference on Computational Complexity*, pages 115–125, 2011.

[Wil13] Ryan Williams. Natural proofs versus derandomization. In *STOC*, pages 21–30, 2013.

[Zha06] Lintao Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *Proceedings of 21st National Conference on Artificial Intelligence and 18th Innovative Applications of Artificial Intelligence Conference (AAAI 2006)*, 2006.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de