Electronic Colloquium on Computational Complexity, Report No. 71 (2012)

A Satisfiability Algorithm and Average-Case Hardness for Formulas over the Full Binary Basis

Kazuhisa Seto
Graduate School of Informatics, Kyoto University
seto@kuis.kyoto-u.ac.jp
Suguru Tamaki
Graduate School of Informatics, Kyoto University
tamak@kuis.kyoto-u.ac.jp

Abstract

We present a moderately exponential time algorithm for the satisfiability of Boolean formulas over the full binary basis. For formulas of size at most cn, our algorithm runs in time $2^{(1-\mu_c)n}$ for some constant $\mu_c > 0$. As a byproduct of the running time analysis of our algorithm, we get strong average-case hardness of affine extractors for linear-sized formulas over the full binary basis.

1 Introduction

In this paper, we are concerned with moderately exponential time algorithms for the Circuit Satisfiability (Circuit SAT) problem. Circuit SAT is, given a Boolean circuit C with n input variables, to determine whether there exists a 0/1 assignment to the input variables such that C outputs 1. It is one of the most fundamental and important NP-complete problems and people have developed many efficient algorithms in both practical and theoretical sense. It is easy to see that one can solve the problem in time $\mathrm{poly}(|C|)2^n$ by brute force search where $|C|$ denotes the size of C. An obvious question is whether there exist moderately exponential time algorithms, i.e., algorithms with the worst case running time of the form $\mathrm{poly}(|C|)2^{(1-\mu)n}$ for some $\mu > 0$.

It is too difficult to answer the above question in affirmative because of the generality of Circuit SAT, that is, many combinatorial problems can be represented as Circuit SAT [4]. Instead of considering Circuit SAT in the most general form, we may investigate the complexity of Circuit SAT over some restricted circuit class $\mathcal{C}$. We write such restricted Circuit SAT as $\mathcal{C}$-SAT. The most well studied restricted circuit class is k-CNF-formulas, which consist of a conjunction of clauses, where each clause is a disjunction of at most k literals. k-CNF-SAT is a central problem in the area of exact exponential algorithms and many efficient algorithms for it have been developed over the past 30 years, see,

ISSN 1433-8092

e.g., [14, 20, 21, 22, 25, 28], and an excellent survey [8]. The best running time upper bound is of the form $\mathrm{poly}(|C|)2^{(1-\mu_k)n}$, where $\mu_k > 0$ is some constant only depending on k. Despite the success of exact algorithms for k-CNF-SAT, there are few works studying the exponential time complexity of Circuit SAT over the more general circuit class until recently.

Let us quickly review some results on $\mathcal{C}$-SAT for more general $\mathcal{C}$. We are aware of the works for CNF-formulas (without restriction on length of each clause) [3, 6, 9, 10, 26, 29] (see also [15]), $\mathbf{AC}^0$ circuits [7, 16], $\mathbf{ACC}^0$ circuits [32], and U_2-formulas (De Morgan formulas) [27]. Here, $\mathbf{AC}^0$ circuits are constant depth circuits over the basis {and, or, not}, where the fan in of each gate is unbounded, $\mathbf{ACC}^0$ circuits are the same as $\mathbf{AC}^0$ circuits except that the basis also contains arbitrary modulo gates of unbounded fan in, and U_2-formulas are formulas over the basis U_2={and, or, not}, where the fan in of {and, or} is two. For CNF-formulas with m clauses, Calabro et al. [6, 8] have shown that Circuit SAT can be solved in time $|C|2^{(1-1/\log(m/n))n}$, for $\mathbf{AC}^0$ circuits of size cn and depth d, Impagliazzo et al. [16] have shown that Circuit SAT can be solved in time $|C|2^{(1-1/O(\log c+d\log d)^{d-1})n}$, for $\mathbf{ACC}^0$ circuits of depth d, Williams [32] has shown that Circuit SAT can be solved in time $|C|2^{n-\Omega(n^{2^{-O(d)}})}$, and for U_2-formulas of size cn, Santhanam [27] have shown that Circuit SAT can be solved in time $|C|2^{(1-1/c^{O(1)})n}$, for example.

In this paper, we extend the result of Santhanam to the case of B_2-formulas which are formulas over the full binary basis B_2 consisting of all two-variable functions. Our main result is the following:

Theorem 1.1. *There is a deterministic algorithm for B_2-formula-SAT which runs in time $2^{(1-\mu_c)n}$ on formulas of size at most cn. Here $\mu_c > 0$ is a constant only depending on c (roughly $\mu_c = 2^{-\Theta(c^3)}$).*

Santhanam's result has an application in proving strong average-case hardness of the parity function against linear-sized U_2-formulas. From the proof of Theorem 1.1, we can show an analogous result, strong average-case hardness of affine extractors against linear-sized B_2-formulas (affine extractors are formally defined in Section 4, see Definition 4.2).

Theorem 1.2. *For any constant $c > 0$, any sequence of B_2-formulas of size at most cn must err in computing affine extractors on at least a $1/2 - 2^{-\Omega(n)}$ fraction of inputs of length n for each n.*

1.1 Background

In this section, we discuss the motivation of designing moderately exponential time algorithms for $\mathcal{C}$-SAT with more general $\mathcal{C}$.

Encoding practical instances

One of the motivations comes from practical applications. Because of its expressibility, Circuit SAT can represent many industrial problems such as soft-

ware and hardware verification and testing, design automation, planning and automated reasoning in a natural way. This motivates the development of faster SAT solvers for instances from practice and today we have very sophisticated SAT solvers which can treat instances of relatively large size. However, most SAT solvers require their input to be in CNF form although natural encoding of industrial problems to Circuit SAT often results in instances represented by general circuits, e.g., formulas with no depth restriction, circuits with parity gates etc. To use fast SAT solvers, first we need to transform the original instances into those in CNF form. After such transformation, the size of instances must increase and in some case the size blow up can be exponential. For example, parity functions have linear size representation in formulas over the full binary basis, but requires quadratic size in de Morgan formulas and exponential size in CNF form. It is more desirable if one can develop efficient algorithms which can treat the original encoding of practical instances.

Proof Techniques

The source of running time savings of SAT algorithms for certain circuit classes such as $\mathbf{AC}^0$ and De Morgan formulas is well explained by the proof technique for circuit lower bounds. That is, $\mathbf{AC}^0$ circuits and De Morgan formulas shrink their sizes well by "random restrictions". Roughly speaking, random restriction chooses, say, $(1-\rho)n$ variables for some $\rho > 0$ randomly and sets random 0/1 values to the chosen variables, and obtains a simplified circuit/formula over the remaining ρn variables. It is well known that by appropriately choosing ρ, random restriction can collapse $\mathbf{AC}^0$ circuits and De Morgan formulas into a constant function with high probability, see, e.g., [1, 11, 12, 34], and [2, 13, 17, 30]. This implies that such circuit classes cannot compute parity functions since such functions remains non-constant functions after random restriction is applied. The above lower bounds argument suggest that backtracking algorithms work well because expected depth of each path in backtracking tree is at most $(1-\rho)n$ with high probability. Unfortunately, for our target class, formulas over the basis {and, or, xor}, random restriction cannot prove interesting lower bounds. It is easy to see that a formula consisting of only xor gates does not shrink into a constant unless we set values to all the variables. To achieve similar savings as Santhanam's result for De Morgan formulas, our algorithm and its analysis require new ideas. We establish a structual result for B_2-formulas of linear size, and use it to solve SAT.

SAT algorithm implies circuit lower bounds

As discussed in the previous paragraph, design and analysis of $\mathcal{C}$-SAT algorithm is often inspired by the corresponding circuit lower bound technique for $\mathcal{C}$. Interestingly, the connection also holds in the reverse order, that is, efficient SAT algorithms implies circuit lower bounds. One of such examples is the result by Paturi et al. [24] showing tight lower bounds for depth three $\mathbf{AC}^0$ circuits computing parity functions. They use a SAT algorithm enumerating solutions of

k-CNF formulas to bound the number of gates in depth three $\mathbf{AC}^0$ circuits. Another example is a recent breakthrough result due to Williams. He proves that SAT algorithm for the circuit class $\mathcal{C}$ with "non-trivial" running time implies that $\mathcal{C}$ does not contain NEXP, the class of languages computable in nondeterministic exponential time [31]. Then he shows non-trivial SAT algorithms for $\mathbf{ACC}^0$ to conclude that $\mathbf{ACC}^0$ does not contain NEXP [32]. Thus, developing SAT algorithms for richer circuit classes is tied to proving lower bounds for those classes.

Strong average-case hardness

We are often interested in proving strong average-case hardness for some circuit class $\mathcal{C}$ rather than just proving worst-case hardness. Here, strong means that any circuit in $\mathcal{C}$ must fail to compute the given function on at least $1/2 - 2^{-\Omega(n)}$ fraction of inputs. Such a hardness result can be used to construct very efficient pseudorandom generators for $\mathcal{C}$. For example, Nisan and Wigderson [23] have shown that if there exists a family of functions computable in $\mathrm{E} = \mathrm{DTIME}(2^{O(n)})$ such that any subexponential size circuits must fail on at least $1/2 - 2^{-\Omega(n)}$ fraction of inputs, then $\mathrm{P} = \mathrm{BPP}$ holds. In general, proving strong (exponential) circuit lower bounds even in the worst-case is a very difficult task for relatively weak circuit classes such as $\mathbf{AC}^0$. However, if we can construct $\mathcal{C}$-SAT algorithms which run in time $2^{n-\Omega(n)}$, they often provide strong average-case lower bounds for $\mathcal{C}$ and actually there are such (but) few results: Calabro et al., Impagliazzo et al. and Lu and Wu [7, 16, 19] and Santhanam [27], respectively show that the parity function is strongly hard for linear-sized $\mathbf{AC}^0$ circuits and U_2 formulas, respectively. Our result adds such rare hardness results in the case of linear-sized B_2-formulas.

Paper organization

In the rest of our paper, we provide detailed algorithms and analysis to support our results. In section 2, we present some useful properties of B_2-formulas, which plays an important role in designing our satisfiability algorithm. In section 3, we give a high level idea, formal description and running time analysis of our algorithm. In section 4, we prove strong average-case hardness results.

2 Preliminaries

Let B_2 be the set of all Boolean functions of two variables. A *B_2-formula* is a rooted binary tree in which each leaf is labeled by a literal from the set $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ or a constant from $\{0, 1\}$ and each internal node is labeled by a function from B_2. Given a B_2-formula ϕ, a *subformula* of ϕ is a B_2-formula which is a subtree in ϕ. By ϕ_v, we denote ϕ's subformula whose root node is v. Every B_2-formula computes in a natural way a Boolean function from $\{0, 1\}^n$ to $\{0, 1\}$. The *size* of a B_2-formula ϕ is defined to be the number of

leaves in it, and it is denoted by $L(\phi)$. We denote by $\mathrm{var}(\phi)$ the set of variables which appear as literals in ϕ. The *frequency* of a variable x in ϕ is defined to be the number of leaves labelled by x or $\overline{x}$, and it is denoted by $\mathrm{freq}_\phi(x)$. We often omit the subscipt ϕ when it is clear from the context. A $\{\wedge, \vee, \oplus\}$-*formula* is a B_2-formula in which each internal node is labeled by $\wedge$ ("and") or $\vee$ ("or") or $\oplus$ ("xor"). It is easy to see that the following holds by using De Morgan's laws and the fact that $\overline{\phi_1 \oplus \phi_2} = \overline{\phi}_1 \oplus \phi_2$.

Fact 2.1. *For any B_2-formula ϕ, there exists a $\{\wedge, \vee, \oplus\}$-formula $\tilde{\phi}$ such that $\tilde{\phi}$ computes the same function as ϕ and $L(\tilde{\phi}) \leq L(\phi)$. Furthermore, we can obtain $\tilde{\phi}$ from ϕ in polynomial time in $L(\phi)$.*

In our SAT algorithm for B_2-formulas, we assume without loss of generality that input formulas are always $\{\wedge, \vee, \oplus\}$-formulas. In what follows, we often write just formula instead of $\{\wedge, \vee, \oplus\}$-formula for brevity. For convenience, we think of constant functions 0 and 1 as formulas.

2.1 Maximal linear nodes

Given a formula ϕ, we define the notion of *linear* node in ϕ inductively as follows. (i) Every leaf is linear. (ii) An internal node is linear if it is labeled by $\oplus$ and its both child nodes are linear. Then, a linear node is *maximal* if its parent node is not linear. Let v be a linear node in ϕ, ϕ_v be the subformula rooted by v and $S = \{y_1, \ldots, y_k\}$ be the set of all leaves in ϕ_v. It is easy to see that ϕ_v computes a linear function $y_1 \oplus \cdots \oplus y_k$. If there exists a variable x which appears in S more than once (as x or $\overline{x}$), ϕ_v is said *redundant*, otherwise it is said *irredundant*. By the commutativity of $\oplus$ and the fact that $0 \oplus y = y$, $1 \oplus y = \overline{y}$, $y \oplus y = 0$ and $y \oplus \overline{y} = 1$ for any literal y, we have:

Fact 2.2. *A redundant subformula ϕ_v can be replaced by an irredundant formula $\tilde{\phi}_v$ which computes the same function as ϕ_v. Note that $L(\tilde{\phi}_v) < L(\phi_v)$ holds and we can obtain $\tilde{\phi}_v$ from ϕ_v in polynomial time in $L(\phi)$.*

The usefulness of the notion of maximal linear nodes is shown by the following lemma.

Lemma 2.3. *Let ϕ be a formula which contains exactly m maximal linear nodes. Then, we can check the satisfiability of ϕ in time $2^m \cdot \mathrm{poly}(L(\phi))$.*

Proof. Let $v_1, \ldots, v_m$ be maximal linear nodes in ϕ. By the inductive definition of maximal linear node, the output of ϕ is fixed when all the outputs of $v_1, \ldots, v_m$ are fixed to constants $a_1, \ldots, a_m \in \{0, 1\}$. There are 2^m possible ways of fixing the outputs of $v_1, \ldots, v_m$. For each possibility, we can check whether there exists a corresponding assignment to $x_1, \ldots, x_n$ or not, by solving a system of linear equations $\phi_{v_1} = a_1, \ldots, \phi_{v_m} = a_m$ in polynomial time using Gaussian elimination. □

The above lemma motivates us to reduce the number of maximal linear nodes in a formula ϕ. We need some definitions. Let u, v be maximal linear nodes in

ϕ. Then u and v is *connected by a $\oplus$-path* if every node in the (unique) path from u to v is labeled by $\oplus$. Such a pair of nodes u, v is called *mergeable* by the following reason. Let s and t be the parent nodes of u and v respectively. Then we can write $\phi_s = \phi_u \oplus \phi_{u'}$ and $\phi_t = \phi_v \oplus \phi_{v'}$ where $\phi_{u'}$ and $\phi_{v'}$ are any formulas. Without loss of generality we can assume $t \neq u'$, and consider the following transformation. First replace ϕ_s by $\tilde{\phi}_s = (\phi_u \oplus \phi_v) \oplus \phi_{u'}$, then replace ϕ_t by $\tilde{\phi}_t = \phi_{v'}$. The resulting formula $\tilde{\phi}$ obviously computes the same function as ϕ by the commutativity of $\oplus$ and we reduce the number of maximal linear nodes by one. Thus we have:

Fact 2.4. *Let ϕ be a formula which contains mergeable pairs of maximal linear nodes. Then there exists a formula $\tilde{\phi}$ which computes the same function as ϕ and does not contain mergeable pairs of maximal linear nodes. Furthermore, $L(\tilde{\phi}) \leq L(\phi)$ and we can obtain $\tilde{\phi}$ from ϕ in polynomial time in $L(\phi)$.*

2.2 Restrictions of $\{\wedge, \vee, \oplus\}$-formulas

For any formula ϕ, any set of variables $\{x_{i_1}, \ldots, x_{i_k}\}$ and any constants $a_1, \ldots, a_k \in \{0, 1\}$, we denote by $\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$ the formula obtained from ϕ by assigning to each x_{i_j}, $\overline{x}_{i_j}$ the value a_j, $\overline{a}_j$ and applying the following procedure **Simplify**.

The procedure **Simplify** reduces the size of a formula applying rules to eliminate constants and redundant literals and gates. It also reduces the number of redundant subformulas and mergeable linear nodes in a formula. These are the same simplification rules used by Håstad [13] and Santhanam [27] with additional rules regarding $\oplus$ gates.

Simplify (ϕ: formula)
Repeat the following until there is no decrease in size of ϕ, number of redundant subformulas and number of mergeable linear nodes.

(a) If $0 \wedge \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by 0.

(b) If $0 \vee \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by ψ.

(c) If $1 \wedge \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by ψ.

(d) If $1 \vee \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by 1.

(e) If $y \vee \psi$ occurs as a subformula, where ψ is a formula and y is a literal, then replace all occurrences of y in ψ by 0 and all occurrence of $\overline{y}$ by 1.

(f) If $y \wedge \psi$ occurs as a subformula, where ψ is a formula and y is a literal, then replace all occurrences of y in ψ by 1 and all occurrence of $\overline{y}$ by 0.

(g) If $0 \oplus \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by ψ.

(h) If $1 \oplus \psi$ occurs as a subformula, where ψ is any formula, replace this subformula by $\overline{\psi}$, where $\overline{\psi}$ denotes a formula which computes the negation of ψ. ($L(\overline{\psi}) \leq L(\psi)$ by Fact 2.1.)

(i) If ψ occurs as a redundant subformula, replace this subformula by an irredundant formula $\tilde{\psi}$ as Fact 2.2.

(j) If ϕ contains mergeable maximal linear nodes, replace ϕ by $\tilde{\phi}$ without them as Fact 2.4.

It is easy to see that **Simplify** runs in time polynomial in the size of ϕ and the resulting formula computes the same function as ϕ. If **Simplify**(ϕ) returns ϕ itself, ϕ is said *irreducible*.

Observation 2.5. *Let ϕ be a formula, x be a variable and $a \in \{0,1\}$ be a constant. Then*

$$L(\phi[x=a]) \leq L(\phi) - \mathrm{freq}(x).$$

Furthermore, if $\mathrm{freq}(x) \geq L(\phi)/|\mathrm{var}(\phi)| + \gamma$ *for some $\gamma \geq 0$, then*

$$L(\phi[x=a]) \leq L(\phi)\left(1 - \frac{1}{|\mathrm{var}(\phi)|}\right)^{1+\gamma|\mathrm{var}(\phi)|/L(\phi)}.$$

Proof. The first inequality is obvious. The following calculation shows the second inequality.

$$\begin{aligned} L(\phi[x=a]) &\leq L(\phi) - L(\phi)/|\mathrm{var}(\phi)| - \gamma \\ &= L(\phi)\left(1 - \frac{1+\gamma|\mathrm{var}(\phi)|/L(\phi)}{|\mathrm{var}(\phi)|}\right) \\ &\leq L(\phi)\left(1 - \frac{1}{|\mathrm{var}(\phi)|}\right)^{1+\gamma|\mathrm{var}(\phi)|/L(\phi)}, \end{aligned}$$

where the last inequality is by $(1-bx) \leq (1-x)^b$ for $0 < b, x < 1$. □

Observation 2.6. *Let ϕ be a formula, v be a node of ϕ where the parent node of v is labeled by $\wedge$ ($\vee$, respectively), and u be a sibling of v. Assume* $\mathrm{var}(\phi_v) = \{x_{i_1}, \ldots, x_{i_k}\}$ *and there exists a variable x which is in* $\mathrm{var}(\phi_u)$ *but not in* $\mathrm{var}(\phi_v)$. *Then for any constants $a_1, \ldots, a_k \in \{0,1\}$ such that $\phi_v[x_{i_1} = a_1, \ldots, x_{i_k} = a_k] = 0$ ($\phi_v[x_{i_1} = a_1, \ldots, x_{i_k} = a_k] = 1$, respectively),*

$$L(\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]) \leq L(\phi) - \left(\sum_{x' \in \mathrm{var}(\phi_v)} \mathrm{freq}(x')\right) - 1.$$

Furthermore, if $\mathrm{freq}(x') \geq L(\phi)/|\mathrm{var}(\phi)|$ *holds for any* $x' \in \mathrm{var}(\phi_v)$*,*

$$\begin{aligned} & L(\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]) \\ \leq\ & L(\phi) \left\{ \prod_{j=0}^{k-2} \left(1 - \frac{1}{|\mathrm{var}(\phi)| - j}\right) \right\} \left(1 - \frac{1}{|\mathrm{var}(\phi)| - k + 1}\right)^{1+|\mathrm{var}(\phi)|/L(\phi)}. \end{aligned}$$

Proof. If we assign 0/1 values to $x_{i_1}, \ldots, x_{i_k}$, obviously $\sum_{x' \in \mathrm{var}(\phi_v)} \mathrm{freq}(x')$ of leaves become constants. Furthermore, by rule (a) of **Simplify** (by rule (d) of **Simplify**, respectively), v's parent node becomes constant. That is, we can remove at least one leaf of ϕ_u whose label is x or $\overline{x}$. The second inequality follows from the first one and using Observation 2.5. □

Observation 2.7. *Let* ϕ *be a formula,* v *be a node of* ϕ *where the parent node of* v *is labeled by* $\wedge$ *or* $\vee$*, and* u *be a sibling of* v*. Assume* $\mathrm{var}(\phi_v) = \{x_{i_1}, \ldots, x_{i_k}\}$ *and* $\mathrm{var}(\phi_v) \supseteq \mathrm{var}(\phi_u)$ *and a variable* x *is in both* $\mathrm{var}(\phi_v)$ *and* $\mathrm{var}(\phi_u)$*. Assume* $x = x_{i_1}$*. Then for any constants* $a_2, \ldots, a_k \in \{0, 1\}$*,*

$$L(\phi[x_{i_2} = a_2, \ldots, x_{i_k} = a_k]) \leq L(\phi) - \left(\sum_{x' \in \mathrm{var}(\phi_v) \setminus \{x_{i_1}\}} \mathrm{freq}(x') \right) - 1.$$

Furthermore, if $\mathrm{freq}(x') \geq L(\phi)/|\mathrm{var}(\phi)|$ *holds for any* $x' \in \mathrm{var}(\phi_v) \setminus \{x_{i_1}\}$*,*

$$\begin{aligned} & L(\phi[x_{i_2} = a_2, \ldots, x_{i_k} = a_k]) \\ \leq\ & L(\phi) \left\{ \prod_{j=0}^{k-3} \left(1 - \frac{1}{|\mathrm{var}(\phi)| - j}\right) \right\} \left(1 - \frac{1}{|\mathrm{var}(\phi)| - k + 2}\right)^{1+|\mathrm{var}(\phi)|/L(\phi)}. \end{aligned}$$

Proof. If we assign 0/1 values to $x_{i_2}, \ldots, x_{i_k}$, obviously $\sum_{x' \in \mathrm{var}(\phi_v) \setminus \{x_{i_1}\}} \mathrm{freq}(x')$ of leaves become constants. Furthermore, ϕ_v becomes x or $\overline{x}$ and ϕ_u becomes one of $0, 1, x, \overline{x}$. That is, by rules (a),(b),(c),(d),(e) and (f),we can remove at least one leaf whose label is x or $\overline{x}$. The second inequality follows from the first one and using Observation 2.5. □

2.3 Useful properties of $\{\wedge, \vee, \oplus\}$-formulas

Our SAT algorithm is a backtracking algorithm which uses four branching rules. The following lemma plays an important role in defining the branching rules of our SAT algorithm.

Lemma 2.8. *In any* n*-variable formula* ϕ *of size* $cn, c \geq 3/4$*, such that* ϕ *does not contain any pair of mergeable maximal linear nodes, at least one of followings holds.*

Case 1: *The number of maximal linear nodes in* ϕ *is less than* $3n/4$*.*

Case 2: *There exists a variable* $x \in \mathrm{var}(\phi)$*,* $\mathrm{freq}(x) \geq c + 1/8c$*.*

Case 3: *There exists a maximal linear node v with $L(\phi_v) \leq 8c$ such that for any $x \in \mathrm{var}(\phi_v), \mathrm{freq}(x) \geq c$ and the parent node of v is labelled by $\wedge$ or $\vee$.*

Proof of Lemma 2.8. Assume that neither Case 1 nor Case 2 occurs. We need the following lemma and fact which are proven later.

Lemma 2.9. *Let ϕ be a formula which contains at least one node labeled by $\wedge$ or $\vee$ but not any pair of mergeable maximal linear nodes. Then, the number of maximal linear nodes whose parent nodes are labeled by $\wedge$ or $\vee$, denoted by $\#\mathrm{MLin}_{\wedge,\vee}(\phi)$, is greater than the number of maximal linear nodes whose parent nodes are labeled by $\{\oplus\}$, denoted by $\#\mathrm{MLin}_{\oplus}(\phi)$.*

Fact 2.10. *(i) The number of maximal linear nodes which have more than $8c$ leaves as descendants is at most $n/8$. (ii) The number of maximal linear nodes which have a variable x with $\mathrm{freq}(x) < c$ is at most $n/8$.*

By Lemma 2.9, $\#\mathrm{MLin}_{\wedge,\vee}(\phi) > (3n/4)/2 = 3n/8$. By Fact 2.10, there are at least $3n/8 - n/8 - n/8 = n/8$ maximal linear nodes satisfying the condition of Case 3.

Proof of Lemma 2.9. We will prove by induction on the size of ϕ.

If $L(\phi) = 1$, then ϕ does not contain a node labeled by $\wedge$ or $\vee$. If $L(\phi) = 2$ or 3, it is easy to check that $\#\mathrm{MLin}_{\wedge,\vee}(\phi) > \#\mathrm{MLin}_{\oplus}(\phi)$ holds.

Now we will show that $\#\mathrm{MLin}_{\wedge,\vee}(\phi) > \#\mathrm{MLin}_{\oplus}(\phi)$ for ϕ whose size is $\ell > 3$ and which contains at least one node labeled by $\wedge$ or $\vee$. If the number of internal nodes which are labeled by $\vee$ or $\wedge$ is exactly one, it is easy to see that $\#\mathrm{MLin}_{\wedge,\vee}(\phi) > \#\mathrm{MLin}_{\oplus}(\phi)$ holds. Thus, assume otherwise. Consider the following two cases. (i) There is a maximal linear node of size at least two. (ii) Every maximal linear node is of size exactly one.

In case (i), pick any maximal linear node of size at least two, say v. Since $L(\phi_v) \geq 2$, ϕ_v contains an internal node w whose child nodes are leaves, say s and t. Let $\tilde{\phi}_v$ be a formula which is identical to ϕ_v except that the nodes s, t are removed from ϕ_v and the node w is replaced by s. Then let $\tilde{\phi}$ be the formula obtained from ϕ by replacing ϕ_v by $\tilde{\phi}_v$. Note that $\tilde{\phi}$ contains at least one node labeled by $\wedge$ or $\vee$ but does not contain any pairs of mergeable maximal linear nodes. Since $L(\tilde{\phi}) = \ell - 2$, it holds that $\#\mathrm{MLin}_{\wedge,\vee}(\tilde{\phi}) > \#\mathrm{MLin}_{\oplus}(\tilde{\phi})$ by the induction hypothesis. It is easy to see that $\#\mathrm{MLin}_{\wedge,\vee}(\phi) = \#\mathrm{MLin}_{\wedge,\vee}(\tilde{\phi})$ and $\#\mathrm{MLin}_{\oplus}(\phi) = \#\mathrm{MLin}_{\oplus}(\tilde{\phi})$ by the construction of $\tilde{\phi}$, we are done.

In case (ii), there exists at least one internal node, say u, whose label is $\vee$ or $\wedge$ and child nodes are leaves, say p and q. Furthermore, there exists at least one more internal node whose label is $\vee$ or $\wedge$. Let v be a parent node of u and w be a sibling of u.

If v is labeled by $\vee$ or $\wedge$, consider a formula $\tilde{\phi}$ obtained from ϕ by replacing u by s. Since $L(\tilde{\phi}) = \ell - 1$, it holds that $\#\mathrm{MLin}_{\wedge,\vee}(\tilde{\phi}) > \#\mathrm{MLin}_{\oplus}(\tilde{\phi})$ by the induction hypothesis. It is easy to see that $\#\mathrm{MLin}_{\wedge,\vee}(\tilde{\phi}) = \#\mathrm{MLin}_{\wedge,\vee}(\phi) - 1$ and $\#\mathrm{MLin}_{\oplus}(\tilde{\phi}) = \#\mathrm{MLin}_{\oplus}(\phi)$, we are done.

If v is labeled by $\oplus$, consider a formula $\tilde{\phi}$ obtained from ϕ by replacing ϕ_v by ϕ_w. It is easy to see that $\#\mathrm{MLin}_{\wedge,\vee}(\phi) - 2 = \#\mathrm{MLin}_{\wedge,\vee}(\tilde{\phi})$ and $\#\mathrm{MLin}_{\oplus}(\phi) = \#\mathrm{MLin}_{\oplus}(\tilde{\phi})$ by the construction of $\tilde{\phi}$, we are done. □

Proof of Fact 2.10. (i) is obvious by the averaging argument. We will show (ii). By the averaging argument, there exists a variable $x \in \mathrm{var}(\phi)$ with $\mathrm{freq}(x) \geq c$. Since there is no variable $x \in \mathrm{var}(\phi)$ with $\mathrm{freq}(x) \geq c + 1/8c$ by assumption, we have $c \leq \mathrm{freq}(x) < c + 1/8c$. Note that if there exists a variable $x' \in \mathrm{var}(\phi)$ with $\mathrm{freq}(x') < c$ then $\mathrm{freq}(x') < c + 1/8c - 1$ because $\mathrm{freq}(x')$ is integer for any variable x'. By the averaging argument, there are at most $n/8c$ variables in $\mathrm{var}(\phi)$ with frequency less than c. The total number of leaves labeled by such variables is at most $c \times n/8c \leq n/8$. □

These proofs complete the proof of Lemma 2.8. □

3 A Satisfiability Algorithm for B_2-Formulas

Before describing our B_2-formula-SAT algorithm and its running time analysis, let us give a basic idea behind them.

Let ϕ be an n-variable formula of size cn. If c is less than $3/4$ or the number of maximal linear nodes in ϕ is less than $3n/4$, then we can check the satisfiability of ϕ in time $2^{3n/4}$. Otherwise, Case 2 or 3 of Lemma 2.8 holds. In such a case, we can reduce the size of ϕ non-trivially by fixing some number of variables to be constants as shown in Observation 2.5, Observation 2.6 and Observation 2.7. Note that if $\mathrm{freq}(x_{i_j}) \geq c$ for any j, then

$$L(\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]) \leq L(\phi) \left\{ \prod_{j=0}^{k-1} \left(1 - \frac{1}{n-j}\right) \right\}$$

holds by repeatedly using Observation 2.5. This decrease of the size from ϕ to $\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$ is called *trivial*. If for some $\gamma > 0$,

$$L(\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]) \leq L(\phi) \left\{ \prod_{j=0}^{k-1} \left(1 - \frac{1}{n-j}\right) \right\} \left(1 - \frac{1}{n}\right)^{\gamma}$$

holds, then the decrease of the size from ϕ to $\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$ is called *non-trivial*. If Observation 2.5 or Observation 2.7 applies to ϕ, $\phi[x = a]$ or $\phi[x_{i_2} = a_2, \ldots, x_{i_k} = a_k]$ is non-trivially reduced for any $a, a_2, \ldots, a_k \in \{0,1\}$. However, if Observation 2.6 applies to ϕ, $\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$ is non-trivially reduced for at least a half fraction of assignments of $a_1, \ldots, a_k \in \{0,1\}$, and $\phi[x_{i_1} = a_1, \ldots, x_{i_k} = a_k]$ is at least trivially reduced for the remaining assignments of $a_1, \ldots, a_k \in \{0,1\}$. To summarize, if we choose certain number of variables appropriately and assign 0/1 values to them uniformly at random, then the formula size non-trivially reduces with probability at least $1/2$. We

would like to estimate the expected size of the reduced formula after assigning values to $(1-\alpha)n$ variables for some $\alpha > 0$.

The lemma described below captures the analysis of the above process. It is a generalization of the Lemma 5 shown in Santhanam's paper [27]. Let $X_0, X_1, \ldots$ be independent random variables which take 0/1 values uniformly at random. Let $\alpha \in (0,1)$ and $\gamma > 0$ be real numbers and β, n be positive integers. We assume n is enough larger than β. Let $Y_n(\alpha, \beta, \gamma)$ be a random variable defined as follows:

$$Y_n(\alpha,\beta,\gamma) := \left\{ \prod_{i=0}^{(1-\alpha)n} \left(1 - \frac{1}{n-i}\right) \right\} \left\{ \prod_{i=0}^{(1-\alpha)n/\beta} \left(1 - \frac{1}{n-\beta i}\right)^{\gamma X_i} \right\}.$$

Lemma 3.1. *For any positive real numbers $\delta \in (0,1)$, $\gamma > 0$, there exist positive real numbers $\alpha \in (0,1)$, $\epsilon > 0$ and a positive integer N, such that for any integer $n \geq N$,*

$$\Pr[Y_n(\alpha,\beta,\gamma) \leq \alpha\delta] \geq 1 - 2^{-\epsilon n}$$

holds.

Proof. In what follows, we ignore integrality issues for simplicity, but the argument holds with slight modification. First note that $\left\{\prod_{i=0}^{(1-\alpha)n}\left(1-\frac{1}{n-i}\right)\right\} \leq \alpha$. Let $\zeta \in (0,1)$ be a small positive real number chosen later. Let I_j be a set of integers defined as

$$I_j := \{(\zeta n)j, \ldots, (\zeta n)(j+1) - 1\}$$

for $0 \leq j \leq \{(1-\alpha)n/\beta + 1\}/\zeta - 1$. It easy to see by the Chernoff bound that

$$\Pr\left[\sum_{i \in I_j} X_i \leq \zeta n/3\right] = 2^{-\Omega_\zeta(n)}$$

for any $j, 0 \leq j \leq \{(1-\alpha)n/\beta + 1\}/\zeta - 1$. Here Ω_ζ hides a constant factor determined by ζ. Thus, we have

$$\Pr\left[\sum_{i \in I_j} X_i > \frac{n}{3} \text{ for any } j, 0 \leq j \leq \frac{(1-\alpha)n/\beta + 1}{\zeta} - 1\right] = 1 - 2^{-\Omega_\zeta(n)}$$

by the union bound. We can show the following fact by an elementary calculation.

Fact 3.2. *If $\sum_{i \in I_j} X_i > \zeta n/3$ for any $j, 0 \leq j \leq \frac{(1-\alpha)n/\beta+1}{\zeta} - 1$, then it holds that*

$$\left\{ \prod_{i=0}^{(1-\alpha)n/\beta} \left(1 - \frac{1}{n-\beta i}\right)^{\gamma X_i} \right\} < (\alpha + \zeta)^{\gamma/3\beta}.$$

Set $\zeta = \alpha$, $2\alpha = \delta^{3\beta/\gamma}$ and choose $\epsilon = \epsilon(\zeta)$ appropriately, we have the desired bound. □

3.1 The Algorithm and Computation Tree

Our satisfiability algorithm for B_2-formulas, **Evalformula** is described in Fig. 1. Without loss of generality, we assume input formulas are irreducible $\{\wedge, \vee, \oplus\}$-formulas. The correctness of **Evalformula** is guaranteed by Lemma 2.8.

EvalFormula (ϕ: Formula, n: integer)
01: /* Case 0 */
02: **if** $L(\phi) = cn < 3n/4$,
03: check the satisfiability of ϕ by brute force search.
04: **if** ϕ is satisfiable, return "yes", **else** return "no".
05: /* Case 1 */
06: **else if** the number of maximal linear nodes is less than $3n/4$,
07: check the satisfiability of ϕ by Lemma 2.3.
08: **if** ϕ is satisfiable, return "yes", **else** return "no".
09: /* Case 2 */
10: **else if** $\exists x \in \text{var}(\phi)$, $\text{freq}(x) \geq c + \frac{1}{8c}$,
11: **EvalFormula**$(\phi[x = 0], n - 1)$,
12: **EvalFormula**$(\phi[x = 1], n - 1)$.
13: /* Case 3 */
14: **else if** $\exists$ maximal linear node v with $L(\phi_v) \leq 8c$
: such that $\forall x \in \text{var}(\phi_v), \text{freq}(x) \geq c$ and
: the parent node of v is labeled by $\wedge$ or $\vee$,
15: assume $\text{var}(\phi_v) = \{x_{i_1}, \ldots, x_{i_k}\}$ and u is a sibling of v.
16: /* Case 3a */
17: **if** $\exists$ x such that $x \in \text{var}(\phi_u)$ and $x \notin \text{var}(\phi_v)$,
18: **for each** constants $a_1, \ldots, a_k \in \{0, 1\}$,
19: **EvalFormula**$(\phi_i[x_{i_1} = a_1, \ldots, x_{i_k} = a_k], n - k)$.
20: /* Case 3b */
21: **else if** $\text{var}(\phi_v) \supseteq \text{var}(\phi_u)$,
22: assume $x_{i_1} \in \phi_u$.
23: **for each** constants $a_2, \ldots, a_k \in \{0, 1\}$,
24: **EvalFormula**$(\phi_i[x_{i_2} = a_2, \ldots, x_{i_k} = a_k], n - k + 1)$.

Figure 1: B_2-formula SAT algorithm

We define a notion of "computation tree" corresponding to the execution of **EvalFormula** on a formula ϕ. A *computation tree* T_ϕ is a binary tree whose nodes are labeled by a triplet $< \psi, s, C >$, where ψ is a formula, s is an integer and C is an element in $\{?, 0, 1, 2, 3a, 3a', 3b, 3b'\}$. ψ, s and C are called *formula label*, (*amortized-*)*size label* and *case label*, respectively. We construct T_ϕ recursively as follows.

Base step: The root node of T_ϕ is labeled by $< \phi, L(\phi), ? >$.

Recursive Step: Let v be a leaf node of T_ϕ whose depth is d and label is $< \psi, s, ? >$.

Case 0: If ψ satisfies the condition of Case 0 in **EvalFormula**, replace v's label by $< \psi, 1, 0 >$.

Case 1: If ψ satisfies the condition of Case 1 in **EvalFormula**, replace v's label by $< \psi, 1, 1 >$.

Case 2: If ψ satisfies the condition of Case 2 in **EvalFormula**, replace v's label by $< \psi, s, 2 >$. Add two nodes v_l, v_r as v's children in the following way:
v_l is labeled by $< \psi[x = 0], L(\psi[x = 0]), ? >$ and
v_r is labeled by $< \psi[x = 1], L(\psi[x = 1]), ? >$.

Case 3a: If ψ satisfies the condition of Cases 3 and 3a in **EvalFormula**, replace v's label by $< \psi, L(\psi), 3a >$. Construct a complete binary tree T_k of height k starting from v as follows.
If a node u is labeled by $< \psi', s, 3a >$ or $< \psi', s, 3a' >$ and at a distance $d' < k$ from v, u's left child is labeled by $< \psi'[x_{i_{d'}} = 0], s - \frac{L(\psi)}{n-d}, 3a' >$ and u's right child is labeled by $< \psi'[x_{i_{d'}} = 1], s - \frac{L(\psi)}{n-d}, 3a' >$.
If a node u is labeled by $< \psi', s, 3a >$ or $< \psi', s, 3a' >$ and at a distance k from v, u's left child is labeled by $< \psi'[x_{i_k} = 0], L(\psi'[x_{i_k} = 0]), ? >$ and u's right child is labeled by $< \psi'[x_{i_k} = 1], L(\psi'[x_{i_k} = 1]), ? >$.

Case 3b: If ψ satisfies the condition of Cases 3 and 3b in **EvalFormula**, replace v's label by $< \psi, L(\psi), 3b >$. Construct a complete binary tree T_{k-1} of height $k-1$ starting from v as follows.
If a node u is labeled by $< \psi', s, 3b >$ or $< \psi', s, 3b' >$ and at a distance $d' < k-1$ from v, u's left child is labeled by $< \psi'[x_{i_{d'+1}} = 0], s - \frac{L(\psi)}{n-d}, 3b' >$ and u's right child is labeled by $< \psi'[x_{i_{d'+1}} = 1], s - \frac{L(\psi)}{n-d}, 3b' >$.
If a node u is labeled by $< \psi', s, 3b >$ or $< \psi', s, 3b' >$ and at a distance $k-1$ from v, u's left child is labeled by $< \psi'[x_{i_k} = 0], L(\psi'[x_{i_k} = 0]), ? >$ and u's right child is labeled by $< \psi'[x_{i_k} = 1], L(\psi'[x_{i_k} = 1]), ? >$.

We will assume that the computation tree is a complete binary tree of depth n by padding it - if there is a node v at depth less than n whose case label is 0 or 1, we add nodes whose labels are $< null, 1, 0 >$ below v.

The following lemma is crucial in the running time analysis of **EvalFormula**. We use the notation $\tilde{L}(p)$ to denote the size label of p in T_ϕ and $\mathrm{d}(p)$ to denote the depth of p in T_ϕ.

Lemma 3.3. *Let p be a node in T_ϕ with $\mathrm{d}(p) \leq n - 8c$ and $T_p(8c)$ be the set of p's descendants which are at distance $8c$ from p. Then, there is a subset $\hat{T}_p(8c)$ of $T_p(8c)$ with $|\hat{T}_p(8c)| = |T_p(8c)|/2$ such that for any $q \in \hat{T}_p(8c)$,*

$$\tilde{L}(q) \leq \max\left\{1, \tilde{L}(p)\left\{\prod_{j=0}^{8c-1}\left(1 - \frac{1}{n - \mathrm{d}(p) - j}\right)\right\}\left(1 - \frac{1}{n - \mathrm{d}(p)}\right)^{1/8c^2}\right\}$$

and for any $q \in T_p(8c) \setminus \hat{T}_p(8c)$,

$$\tilde{L}(q) \leq \max\left\{1, \tilde{L}(p)\left\{\prod_{j=0}^{8c-1}\left(1 - \frac{1}{n - \mathrm{d}(p) - j}\right)\right\}\right\}.$$

Proof. If the case label of p is 2, 3a or 3b, we obtain the desired bound from Observations 2.5, 2.6 or 2.7 respectively according to the case label of p. If the case label of p is 3a' or 3b', we can also use Observations 2.6 or 2.7 respectively because of the definition of amortized-size. □

3.2 Running time analysis

Let $T_\phi(d)$ denote the set of depth d nodes in T_ϕ. For $p \in T_\phi$, P_p denotes the path from the root of T_ϕ to p and $P_p(d)$ denotes the node of P_p with depth d. We define a function $X_i(p)$ from $T_\phi((1-\alpha)n)$ to $\{0,1\}$ as follows:

$$X_i(p) := \begin{cases} 1 & \text{if } P_p(8c(i+1)) \in \hat{T}_{P_p(8ci)}(8c) \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 3.4. *Let p be drawn from $T_\phi((1-\alpha)n)$ uniformly at random. Then*

$$\Pr[X_0(p) = a_0, \ldots, X_{(1-\alpha)n/8c}(p) = a_{(1-\alpha)n/8c}] = 1/2^{(1-\alpha)n/8c+1}$$

for any $a_0, \ldots, a_{(1-\alpha)n/8c} \in \{0,1\}$. That is, $X_0, \ldots, X_{(1-\alpha)n/8c}$ are independent random variables which take values 0 and 1 uniformly at random.

Proof. It follows from that we define $\hat{T}_q(8c)$ as $|\hat{T}_q(8c)| = |T_q(8c)|/2$ in Lemma 3.3. □

Lemma 3.5. *Let p be a node of $T_\phi((1-\alpha)n)$. Then*

$$\tilde{L}(p) \leq \max\{1, L(\phi)Y_n(\alpha, 8c, 1/8c^2)(p)\}.$$

where

$$Y_n(\alpha,\beta,\gamma)(p) := \left\{\prod_{i=0}^{(1-\alpha)n}\left(1 - \frac{1}{n-i}\right)\right\}\left\{\prod_{i=0}^{(1-\alpha)n/\beta}\left(1 - \frac{1}{n-\beta i}\right)^{\gamma X_i(p)}\right\}.$$

Proof. It follows from Lemma 3.3 and the definition of $X_i(p)$. □

Combining Lemma 3.1 and 3.5, we have:

Lemma 3.6. *Let p be drawn from $T_\phi((1-\alpha)n)$ uniformly at random. Then*

$$\Pr[\tilde{L}(p) < 3\alpha n/4] = 1 - 2^{-\Omega_\alpha(n)}$$

for $\alpha = (3/4)^{192c^3}/2$.

Now we are ready to prove Theorem 1.1.

Proof of Theorem 1.1. Let ϕ be an n-variable formula and consider a corresponding computation tree T_ϕ. We upper bound the running time of **EvalFromula**(ϕ,n) by the sum of the following four values:
(i) Let $N_0(d)$ be the number of nodes whose depth is d and case label is 0. Define T_0 as

$$T_0 := \sum_{d=0}^{(1-\alpha)n} N_0(d) \cdot 2^{3(n-d)/4}.$$

(ii) Let $N_1(d)$ be the number of nodes whose depth is d and case label is 1. Define T_1 as

$$T_1 := \sum_{d=0}^{(1-\alpha)n} N_1(d) \cdot 2^{3(n-d)/4}.$$

(iii) Let $N_{2,3}$ be the number of nodes whose depth is $(1-\alpha)n$ and size label is less than $3\alpha n/4$. Define $T_{2,3}$ as

$$T_{2,3} := N_{2,3} \cdot 2^{3\alpha n/4}.$$

(iv) Let $N'_{2,3}$ be the number of nodes whose depth is $(1-\alpha)n$ and size label is at least $3\alpha n/4$. Define $T'_{2,3}$ as

$$T'_{2,3} := N'_{2,3} \cdot 2^{\alpha n}.$$

It is easy to see that $\mathrm{poly}(L(\phi))(T_0+T_1+T_{2,3}+T'_{2,3})$ upper bounds the running time of **EvalFromula**(ϕ,n). $N_0(d), N_1(d)$ is at most 2^d and $N_{2,3}$ is at most $2^{(1-\alpha)n}$. Lemma 3.6 shows that $N'_{2,3} \leq 2^{(1-\alpha)n} \times 2^{-\Omega(n)}$. Therefore, $T_0 + T_1 + T_{2,3} + T'_{2,3} = 2^{n-\Omega(n)}$. □

4 Strong average-case hardness

We will show strong average-case hardness of affine extractors for linear-sized $\{\wedge, \vee, \oplus\}$-formulas, as claimed in Theorem 1.2. At first, we give definitions of affine source and affine extractor.

Definition 4.1. *Let F_2 be the finite field with 2 elements. Denote by F_2^n the n dimensional vector space over F_2. A distribution X over F_2^n is an (n,k)-affine source if there exist linearly independent vectors $a_1, \cdots, a_k \in F_2^n$ and another vector $b \in F_2^n$ such that X is sampled by choosing $x_1, \cdots, x_k \in F$ uniformly and independently and computing*

$$X = \sum_{i=1}^{k} x_i a_i + b.$$

An affine extractor is a deterministic function such that given any affine source as the input, the output of the function is statistically close to the uniform distribution.

Definition 4.2. *A function $f_n : F_2^n \to F_2$ is a deterministic (k, ϵ)-affine extractor if for every (n, k)-affine source X,*

$$1/2 - \epsilon \leq \Pr[f_n(X) = 0] \leq 1/2 + \epsilon.$$
$$(\textit{And } 1/2 - \epsilon \leq \Pr[f_n(X) = 1] \leq 1/2 + \epsilon.)$$

We need the following Theorem due to [5, 18, 33].

Theorem 4.3. *For every $\delta > 0$ there exists a polynomial time computable family of (k, ϵ)-extractors $\mathbf{AE}_n : F_2^n \to F_2$ with $k = \delta n$ and $\epsilon = 2^{-\Omega_\delta(n)}$.*

Proof of Theorem 1.2. Let ϕ be an n-variable formula of size at most cn and consider the computation tree T_ϕ constructed by **EvalFormula** on ϕ. Here we treat T_ϕ as the original one, i.e., we do not add any nodes to make T_ϕ a complete binary tree of height n. Let $\tilde{T}_\phi$ be the set of all leaf nodes in T_ϕ. Each node p of $\tilde{T}_\phi$ defines a subcube $C(p)$ of $\{0,1\}^n$. Note that any node p in $\tilde{T}_\phi$ has case label 0 or 1.

Fact 4.4. *If p has case label 0 and is at depth d, $C(p)$ can be partitioned into the set of subcubes $\{C_1, \ldots, C_k\}, k \leq 2^{3(n-d)/4}$ such that each C_i has dimension at least $(n-d)-3(n-d)/4 = (n-d)/4$ and ϕ becomes constant on C_i. Furthermore, if $d \leq (1-\alpha)n$,*

$$\Pr_{x \in C_i}[\phi(x) = \mathbf{AE}_n(x)] \leq 1/2 + 1/2^{\Omega(n)}.$$

Fact 4.5. *If p has case label 1 and is at depth d, $C(p)$ can be partitioned into the set of affine subspaces $\{C_1, \ldots, C_k\}, k \leq 2^{3(n-d)/4}$ such that each C_i has dimension at least $(n-d) - 3(n-d)/4 = (n-d)/4$ and ϕ becomes constant on C_i. Furthermore, if $d \leq (1-\alpha)n$,*

$$\Pr_{x \in C_i}[\phi(x) = \mathbf{AE}_n(x)] \leq 1/2 + 1/2^{\Omega(n)}.$$

In the proof of Theorem 1.1, we can see that

$$\Pr_{x \in \{0,1\}^n}\left[x \in \bigcup_{p \in \tilde{T}_\phi : \mathrm{d}(p) > (1-\alpha)n} C(p)\right] \leq 2^{-\Omega(n)}.$$

Therefore,

$$\begin{aligned}\Pr_x[\phi(x) = \mathbf{AE}_n(x)] &= \sum_{p \in \tilde{T}_\phi} \Pr_{x \in \{0,1\}^n}[x \in C(p)] \Pr_{x \in C(p)}[\phi(x) = \mathbf{AE}_n(x)] \\ &= \sum_{\substack{p \in \tilde{T}_\phi \\ \mathrm{d}(p) \leq (1-\alpha)n}} \Pr_{x \in \{0,1\}^n}[x \in C(p)] \Pr_{x \in C(p)}[\phi(x) = \mathbf{AE}_n(x)] \\ &\quad + \sum_{\substack{p \in \tilde{T}_\phi \\ \mathrm{d}(p) > (1-\alpha)n}} \Pr_{x \in \{0,1\}^n}[x \in C(p)] \Pr_{x \in C(p)}[\phi(x) = \mathbf{AE}_n(x)]\end{aligned}$$

and

$$\sum_{\substack{p \in \tilde{T}_\phi \\ \mathrm{d}(p) \leq (1-\alpha)n}} \Pr_{x \in \{0,1\}^n}[x \in C(p)] \Pr_{x \in C(p)}[\phi(x) = \mathbf{AE}_n(x)] \leq 1/2 + 2^{-\Omega(n)}$$

by Facts 4.4 and 4.5 , and

$$\sum_{\substack{p \in \tilde{T}_\phi \\ \mathrm{d}(p) > (1-\alpha)n}} \Pr_{x \in \{0,1\}^n}[x \in C(p)] \Pr_{x \in C(p)}[\phi(x) = \mathbf{AE}_n(x)] \leq 2^{-\Omega(n)}.$$

This completes the proof. □

Acknowledgment

The authors are deeply grateful to Alexander S. Kulikov for useful discussion and helpful comments on the manuscript.

References

[1] Miklós Ajtai. Σ^1_1-formulae on finite structures. Ann. Pure Appl. Logic, 24(1):1–48, 1983.

[2] A. E. Andreev. On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes. Moscow Univ. Math. Bull., 42(1):63–66, 1987.

[3] Vikraman Arvind and Rainer Schuler. The Quantum Query Complexity of 0-1 Knapsack and Associated Claw Problems. In Proceedings of the 14th Annual International Symposium on Algorithm and Computation (ISAAC), LNCS 2906, pages 168–177, 2003.

[4] A. Biere, M. Heule, H. van Maaren and T. Walsh, editors. Handbook of Satisfiability. IOS Press, 2008.

[5] Jean Bourgain. On the construction of affine-source extractors. Geometric and Functional Analysis, 17(1):33–57, 2007.

[6] Chris Calabro, Russell Impagliazzo and Ramamohan Paturi. A Duality between Clause Width and Clause Density for SAT. In Proceedings of the 21st Annual IEEE Conference Computational Complexity (CCC), pages 252–260, 2006.

[7] Chris Calabro, Russell Impagliazzo and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC), pages 75–85, 2009.

[8] Evgeny Dantsin and Edward Hirsch. Worst-case upper bounds. In A. Biere, M. Heule, H. van Maaren and T. Walsh, editors, Handbook of Satisfiability, IOS Press, 2008.

[9] Evgeny Dantsin, Edward A. Hirsch and Alexander Wolpert. Algorithms for SAT Based on Search in Hamming Balls. In Proceedings of the 21st Annual Symposuim on Theoretical Aspects of Computer Science (STACS), LNCS 2996, pages 141–151, 2004.

[10] Evgeny Dantsin, Edward A. Hirsch and Alexander Wolpert. Clause Shortening Combined with Pruning Yields a New Upper Bound for Deterministic SAT Algorithms. In Proceedings of the 6th Conference on Algorithms and Complexity (CIAC), LNCS 3998, pages 60–68, 2006.

[11] M. L. Furst, J. B. Saxe and M. Sipser. Parity, circuits, and the polynomialtime hierarchy. Math. Systems Theory, 17:13–27, 1984.

[12] Johan Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. In Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC), pages 6–20, 1986.

[13] Johan Håstad. The shrinkage exponent of De Morgan formulas is 2. SIAM Journal on Computing, 27(1):48–64, 1998.

[14] Timon Hertli. 3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General. In Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), pages 277–284 , 2011.

[15] Edward A. Hirsch. Exact Algorithms for General CNF SAT. In Encyclopedia of Algorithms, Springer, 2008.

[16] Russell Impagliazzo, William Matthews and Ramamohan Paturi. A Satisfiability Algorithm for AC^0. In Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 961–972, 2012.

[17] Russell Impagliazzo and Noam Nisan. The Effect of Random Restrictions on Formula Size. Random Struct. Algorithms, 4(2):121–134, 1993.

[18] Xin Li. A New Approach to Affine Extractors and Dispersers. In Proceedings of the 26th Annual IEEE Conference on Computational Complexity (CCC), pages 137–147, 2011.

[19] Chi-Jen Lu and Hsin-Lung Wu. On the Hardness against Constant-Depth Linear-Size Circuits. In Proceedings of the 16th Annual International Computings and Combinatorics Conference (COCOON), LNCS 6196, pages 13–22, 2010.

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

[21] Robin A. Moser and Dominik Scheder. A full derandomization of Schöning's k-SAT algorithm. In Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC), pages 245–252, 2011.

[22] Kazuhisa Makino, Suguru Tamaki, Masaki Yamamoto. Derandomizing HSSW Algorithm for 3-SAT. In Proceedings of the 17th Annual International Computings and Combinatorics Conference (COCOON), LNCS 6842, pages 1–12, 2011.

[23] Noam Nisan and Avi Wigderson. Hardness vs Randomness. J. Comput. Syst. Sci., 49(2):149–167, 1994.

[24] Ramamohan Paturi, Pavel Pudlák and Francis Zane. Satisfiability Coding Lemma. Chicago J. Theor. Comput. Sci., 1999.

[25] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks and Francis Zane. An improved exponential-time algorithm for k-SAT. J. ACM, 52(3):337–364, 2005.

[26] Pavel Pudlák. Satisfiability - Algorithms and Logic. In Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS 1450, pages 129–141, 1998.

[27] Rahul Santhanam. Fighting Perebor: New and Improved Algorithms for Formula and QBF Satisfiability. In Proceedings of the 51st International Symposium on Foundations of Computer Science (FOCS), pages 183–192, 2010.

[28] Uwe Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS), pages 410–414, 1999.

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

[30] B. A. Subbotovskaya. Realizations of linear functions by formulas using and, or, not. Soviet Mathematics Doklady, 2:110–112, 1961.

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

[32] Ryan Williams. Non-uniform ACC Circuit Lower Bounds. In Proceedings of the 26th Annual IEEE Conference on Computational Complexity (CCC), pages 115–125, 2011.

[33] Amir Yehudayoff. Affine extractors over prime fields. Combinatorica, 31(2):245–256, 2011.

[34] Andrew Chi-Chih Yao. Separating the Polynomial-Time Hierarchy by Oracles (Preliminary Version). In Proceedings of the 26th International Symposium on Foundations of Computer Science (FOCS), pages 1–10, 1985.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de