2016 IEEE 57th Annual Symposium on Foundations of Computer Science

A better-than-3n lower bound for the circuit complexity of an explicit function

Magnus Gausdal Find*, Alexander Golovnev†‡, Edward A. Hirsch‡ and Alexander S. Kulikov‡
*National Institute of Standards and Technology, USA
†New York University, USA
‡Steklov Institute of Mathematics at St. Petersburg, Russian Academy of Sciences, Russia

***Abstract*—We consider Boolean circuits over the full binary basis. We prove a $(3 + \frac{1}{86})n - o(n)$ lower bound on the size of such a circuit for an explicitly defined predicate, namely an affine disperser for sublinear dimension. This improves the $3n - o(n)$ bound of Norbert Blum (1984). The proof is based on the gate elimination technique extended with the following three ideas. We generalize the computational model by allowing circuits to contain cycles, this in turn allows us to perform affine substitutions. We use a carefully chosen circuit complexity measure to track the progress of the gate elimination process. Finally, we use quadratic substitutions that may be viewed as delayed affine substitutions.**

***Keywords*-Affine dispersers; Boolean circuits; lower bounds.**

I. Introduction

In this paper we consider Boolean circuits over the full binary basis. A simple counting argument [1] shows that most Boolean functions require circuits of exponential size. However, showing superpolynomial lower bounds for explicitly defined functions (for example, for functions from $\mathbf{NP}$) remains a hopelessly difficult task. (In particular, such lower bounds would imply $\mathbf{P} \neq \mathbf{NP}$.) Moreover, even superlinear bounds are unknown for functions in $\mathbf{E}^{\mathbf{NP}}$. Superpolynomial bounds are known for $\mathbf{MAEXP}$ (exponential-time Merlin-Arthur games) [2], and arbitrary polynomial lower bounds are known for $\mathbf{O_2}$ (the oblivious symmetric second level of the polynomial hierarchy) [3].

People started to tackle the problem in the 60s. Kloss and Malyshev [4] proved a lower bound of $2n - O(1)$. Schnorr [5] proved a $2n - O(1)$ lower bound for a class of functions with certain structure. Stockmeyer [6] proved a $2.5n - O(1)$ bound for certain symmetric functions. Paul [7] proved a $2.5n - o(n)$ lower bound for a variant of the storage access functions. Eventually, Blum [8] extended Paul's argument and proved a $3n - o(n)$ bound.

Blum's bound remained unbeaten for more than thirty years. By reviewing the proof, one notes that it cannot be extended to get a stronger than $3n$ lower bound without using different properties of functions.

Recently, Demenkov and Kulikov [9] presented a much simpler proof of a $3n-o(n)$ lower bound for functions with an entirely different property: affine dispersers. This property allows to make affine substitutions until the disperser's dimension is reached. As was later noted by Vadhan and Williams [10], the way Demenkov and Kulikov use this property cannot give stronger than $3n$ bounds as it is tight for the inner product function (which is known to be an affine disperser for dimension $n/2+1$). Hence, mysteriously, two different proofs using two different properties are both stuck on exactly the same lower bound $3n - o(n)$ which was first proved more than 30 years ago. Is this lack of progress grounded in combinatorial properties of circuits and this line of research faces an insurmountable obstacle? Or can refinements on known techniques go above $3n$? In this paper we show that the latter is the case. We eventually improve the bound for affine dispersers to $(3 + \frac{1}{86})n - o(n)$, which is stronger than Blum's bound.

In this paper we eventually improve the bound for affine dispersers to $(3+\frac{1}{86})n-o(n)$, which is stronger than Blum's bound.

Other models: The exact complexity of computational problems is different in different models of computation: for example, switching from multitape to single-tape Turing machines squares the time complexity; random access machines are even more efficient. Boolean circuits over the full binary basis make a very robust computational model. Using a different constant-arity basis only changes the constants in the complexity. A fixed set of gates of arbitrary arity (for example, ANDs, ORs and XORs) still preserves the complexity in terms of the number of wires. After all, finding a function hard for Boolean circuits can be viewed as a combinatorial problem, in a contrast to lower bounds for uniform models. Therefore, breaking the linear barrier for Boolean circuits can be viewed as an important milestone on the way to stronger complexity lower bounds.

Stronger than $3n$ lower bounds are known for various restricted bases. One of the most popular such bases, U_2, consists of all binary Boolean functions except for parity (xor) and its negation (equality), Schnorr [11] proved that the circuit complexity of the parity function is $3n-3$. Zwick [12] gave a $4n - O(1)$ lower bound for certain symmetric functions, Lachish and Raz [13] showed a $4.5n-o(n)$ lower bound for an $(n - o(n))$-mixed function (a function all of whose subfunctions of any $n - o(n)$ variables are different). Iwama and Morizumi [14] improved this bound to $5n-o(n)$. Demenkov et al. [15] gave a simpler proof of a $5n - o(n)$ lower bound for a function with $o(n)$ outputs as well as presented a $7n - o(n)$ lower bound for a function with n outputs. It is interesting to note that the progress on U_2

0272-5428/16 $31.00 © 2016 IEEE
DOI 10.1109/FOCS.2016.19

circuit lower bounds is also stuck on the $5n - o(n)$ lower bound: Amano and Tarui [16] presented an $(n-o(n))$-mixed function whose circuit complexity over U_2 is $5n + o(n)$.

It was recently showed that depth 2 circuits with unbounded fanin $\wedge$, $\oplus$-gates cannot compute affine dispersers with good parameters [17].

While we do not have nonlinear bounds for constant-arity Boolean circuits, stronger bounds are known for weaker models, including monotone circuits (Razborov [18]), circuits of constant depth with no XOR (Yao and Håstad [19], [20]), circuits of polylogarithmic depth over infinite fields (Shoup and Smolensky [21]), formulas (Subbotovskaya [22], Khrapchenko [23], [24], Andreev [25], Impagliazzo and Nisan [26], Paterson and Zwick [27], Håstad [28] and Tal [29]). These bounds, however, do not translate to superlinear lower bounds for general constant-arity Boolean circuits.

Connections to CircuitSAT algorithms: A recent promising direction initiated by Williams [30] connects the complexity of circuits to the complexity of algorithms for CircuitSAT (this is the problem of checking whether a given circuit has a satisfying assignment, that is, a substitution of inputs by constants that forces the circuit to output one). Namely, the existence of better-than-2^n algorithms for CircuitSAT for a particular circuit model implies exponential lower bounds for these circuits for functions in large classes like $\mathbf{NEXP}$. This way unconditional exponential lower bounds have been proved for ACC_0 circuits (constant-depth circuits with unbounded-arity OR, AND, NOT, and arbitrary modular gates) [31]. Ben-Sasson and Viola [32] have demonstrated that in order to prove a specific linear lower bound for a function in $\mathbf{E}^{\mathbf{NP}}$ it suffices to lower the base of the exponent in the 3-SAT complexity down to an appropriate constant. It should be noted, however, that currently available algorithms for the satisfiability problem for general circuit classes are not sufficient for proving new lower bounds.

Also techniques similar to the ones used in proving circuit lower bounds algorithms are employed in a number of algorithms for CircuitSAT and FormulaSAT, see e.g. [33], [34], [35], [36], [37], [38], [39].

Our methods: Almost all previous lower bounds have been proved using a simple gate elimination technique: one gradually simplifies the function (for example, by substituting variables one by one) showing that every simplification step eliminates a certain number of gates. A crucial idea [5] is to keep the function in the same class. Following [9], we prove lower bounds for affine dispersers, that is, functions that are non-constant on affine subspaces of certain dimensions: Ben-Sasson and Kopparty [40] gave an explicit construction of affine dispersers for sublinear dimensions.

Feeding an appropriate constant to a non-linear gate (for example, AND) makes this gate constant and therefore eliminates subsequent gates, which helps to eliminate more gates than in the case of a linear gate (for example, XOR). On the other hand, linear gates, when stacked together, sometimes allow to reorganize the circuit. Then affine restrictions can kill such gates while keeping the properties of an affine disperser. This idea has been used in [7], [6], [8], [36], [9].

Thus, it is natural to consider a circuit as composed of linear circuits connected by non-linear gates. In our case analysis we make affine substitutions but not restrictions. That is, instead of just saying that $x_1 \oplus x_2 \oplus x_3 \oplus x_9 = 0$ and removing all gates that become constant, we make sure to replace all occurrences of x_1 by $x_2 \oplus x_3 \oplus x_9$. Since a gate computing such a sum might be unavailable and we do not want to increase the number of gates, we "rewire" some parts of the circuit, which, however, may potentially introduce cycles. This is the first ingredient of our proof: *cyclic circuits*. That is, the linear components of our "circuits" may now have directed cycles; however, we require that the values computed in the gates are still uniquely determined. Cyclic circuits have already been considered in [41], [42], [43], [44] (the last reference contains an overview of previous work on cyclic circuits).

Thus we are able to make affine substitutions. We try to make such a substitution in order to make the topmost (i.e., closest to the inputs) non-linear gate constant. This, however, does not seem to be enough. The second ingredient in our proof is a *complexity measure* that manages difficult situations (bottlenecks) by allowing to perform an amortized analysis: we count not just the number of gates, we compute a linear combination of the number of gates and the number of bottlenecks. Such measures were previously considered by several authors. For example, Zwick [12] counted the number of (internal) gates minus the number of inputs of outdegree 1. The same measure was later used by Lachish and Raz [13] and by Iwama and Morizumi [14]. Kojevnikov and Kulikov [45] used a measure assigning different weights to linear and non-linear gates to show that Schnorr's $2n - O(1)$ lower bound [11] can be strengthened to $7n/3 - O(1)$. Carefully chosen complexity measures are also used to estimate the progress of splitting algorithms for **NP**-hard problems [46], [47], [48].

Our main bottleneck (called "troubled gate") is a gate of outdegree 1 that is fed by two inputs x and y of degree 2, and that computes $(x \oplus a)(y \oplus b) \oplus c$ for some constants $a, b, c \in \{0, 1\}$.

Sometimes in order to fight a troubled gate, we have to make a *quadratic substitution*, which is the third ingredient of our proof. This happens if the gate below G is a linear gate fed by a variable z; in the simplest case a substitution $z = xy$ kills G, the linear gate, and the gate below (actually, we show it kills much more). However, quadratic substitutions may make affine dispersers constant, so we consider a special type of quadratic substitutions. Namely, we consider quadratic substitutions as a form of delayed affine substitutions (in the example above, if we promise to substitute later a constant either to x or to y, the substitution can be considered affine). In order to maintain this, instead of affine subspaces (where affine dispersers are non-constant by definition) we consider so-called read-once depth-2 quadratic sources (essentially, this means that all variables in the right-hand sides of the quadratic substitutions that we make are pairwise distinct free variables). We show that an affine disperser for a sublinear dimension remains non-constant for read-once depth-2 quadratic sources of a sublinear dimension.

Open questions and further applications of the methods: A natural further direction is to apply the developed techniques (quadratic substitutions, cyclic circuits, and combined complexity measures) to get new complexity lower bounds and satisfiability upper bounds for other circuit models.

An affine disperser for dimension d may be viewed as a function that is not constant on any affine subspace of *size* at least 2^d. A natural extension is a function that is not constant on similarly sized varieties defined by quadratic polynomials. Golovnev and Kulikov [49] presented a short proof that such "quadratic dispersers" with appropriate parameters must have circuit size at least $3.1n$. However, explicit constructions of such dispersers are currently unknown. There are known constructions of dispersers for algebraic varieties over large finite fields [50], and known constructions of such dispersers over $\mathbb{F}_2$ [17], [51] but with weaker parameters than needed for the lower bound to work.

Using quadratic substitutions and combined complexity measures, Golovnev et al. [52] recently improved known upper bounds for satisfiability algorithms for general circuits as well as average case circuit size lower bounds.

It would be useful to understand the limitations of the method used in this paper. Do there exist affine dispersers for sublinear dimension computable by circuits of linear size? More generally, is there an inherent limitation of the gate elimination technique, that is, can it give a non-linear or arbitrary linear lower bound in principle?

II. Definitions

Gates and notation: A circuit is an acyclic directed graph in which incoming edges are numbered for every node. The nodes are called *gates*. A gate may have either indegree zero (in which case it is called an *input* gate, or a *variable*) or indegree two (in which case it is called an *internal* gate). Every internal gate is labelled by a Boolean function $g\colon \{0,1\} \times \{0,1\} \to \{0,1\}$, and the set of all the sixteen such functions is denoted by B_2. We call these binary functions *operations* in order to distinguish them from functions of n variables computed in the gates. The size of a circuit is the number of internal gates.

We say that an operation is of *and-type* if it computes $g(x,y) = (c_1 \oplus x)(c_2 \oplus y) \oplus c_3$ for some constants $c_1, c_2, c_3 \in \{0,1\}$, and of *xor-type* if it computes $g(x,y) = x \oplus y \oplus c_1$ for some constant $c_1 \in \{0,1\}$. Similarly, we call gates and-type and xor-type. If a gate computes an operation depending on precisely one of its inputs, we call it *passing*.

If an (internal) gate computes a constant operation, we call it *trivial* (note that it still has two incoming edges). If a substitution forces some gate G to compute a constant, we say that it *trivializes* G. (For example, for a gate computing the operation $g(x,y) = x \wedge y$, the substitution $x = 0$ trivializes it.)

We denote by $out(G)$ the outdegree of the gate G. If $out(G) = k$, we call G a k-gate. If $out(G) \geq k$, we call it a k^+-gate. We adopt the same terminology for variables (so we have 0-variables, 1-variables, 2^+-variables, etc.).

One gate of outdegree zero is designated as the output.

An *affine disperser* for dimension $d(n)$ is a family of functions $f_n\colon \mathbb{F}_2^n \to \mathbb{F}_2$ such that for all sufficiently large n, f_n is non-constant on any affine subspace of dimension at least $d(n)$. Explicit constructions of affine dispersers have drawn a lot of attention recently. First, polynomial-time computable affine dispersers for any linear dimension were constructed [53], [54], and then it was shown that there are polynomial-time computable affine dispersers for sublinear dimensions $d(n) = o(n)$ [40], [55], [56], [51], [57].

A. Generalizations of circuits

Cyclic circuits: In this paper we apply a sequence of transformations on circuits. To accomodate this we use a generalizations of circuits. These generalized circuits may contain cycles of a certain kind; however we only introduce cycles in such a way that the values computed in the gates are internally consistent.

A *cyclic circuit* is a directed (not necessarily acyclic) graph where all vertices have indegree either 0 or 2. We adopt the same terminology for its nodes (input and internal gates) and its size as for ordinary circuits. We restrict our attention to *cyclic xor-circuits*, where all gates compute affine operations. While the most interesting internal gates compute either $\oplus$ or $\equiv$, for technical reasons we also allow passing gates and trivial gates. We will be interested in multioutput cyclic circuits, so, in contrast to our definition of ordinary circuits, several gates may be designated as outputs, and they may have nonzero outdegree.

A circuit, and even a cyclic circuit, naturally corresponds to a system of equations over $\mathbb{F}_2$. Variables of this system correspond to the values computed in the internal gates. The operation of an internal gate imposes an equation defining the computed value. Whenever an input gate is encountered, it is treated like a constant (because we will be interested in solving this system when we are given specific input values). Thus we formally have a separate system for every assignment to the input gates, but all these systems share the same matrix. For a gate G fed by gates F and H and computing some operation $\odot$, we write the equation $G \oplus (F \odot H) = 0$. A more specific clarifying example would be a gate G computing $F \oplus x \oplus 1$, where x is an input gate; then the line in the system would be $G \oplus F = x \oplus 1$, where G and F contribute two 1's to the matrix, and $x \oplus 1$ contributes to the constant vector.

For a cyclic xor-circuit, this is a linear system with a square matrix. We call a cyclic xor-circuit *fair* if this matrix has full rank. It follows that for every assignment of the inputs, there exist *unique* values for the gates such that these values are consistent with the circuit (that is, for each gate its value is correctly computed from the values in its inputs). Thus, similarly to an ordinary circuit, every gate in a fair circuit computes a function of the values fed into its input gates (clearly, it is an affine function). Note that if we additionally impose the requirement that the graph is acyclic, we arrive at ordinary linear circuits (that is, circuits consisting of xor-type gates, passing gates, and constant gates).

Semicircuits: We introduce the following notion, called *semicircuits*, a generalization of both Boolean circuits and cyclic xor-circuits.

A semicircuit is a composition of a cyclic xor-circuit and an (ordinary) circuit. Namely, its nodes can be split into two sets, X and C. The nodes in the set X form a cyclic xor-circuit. The nodes in the set C form an ordinary circuit (if wires going from X to C are replaced by variables). There are no wires going back from C to X. A semicircuit is called fair if X is fair. ***In what follows we abuse the notation by using the word "circuit" to mean a fair semicircuit.***

III. LOWER BOUND

A. Overview

Section III is devoted to the proof of the main theorem.

The proof goes by induction. We start with an affine disperser and a circuit computing it on $\{0,1\}^n$. Then we gradually shrink the space where it is computed by adding equations ("substitutions") for variables. This allows us to simplify the circuit by reducing the number of gates (and other parameters counted in the complexity measure) and eliminating the variable we have just substituted.

In Section III-B we show how to make substitutions in fair semicircuits, and how to normalize them afterwards. We introduce five normalization rules covering various degenerate cases that may occur in a circuit after applying a substitution to it: e.g., a gate of outdegree 0, a gate computing a constant function, a gate whose value depends on one of its inputs only. For each such case, we show how to simplify a circuit.

We then show how to make affine substitutions. This is the step that might potentially introduce cycles in the affine part of a circuit and that requires to work with a generalized notion of circuits.

Also, we define a so-called troubled gate. Informally speaking, this is a special bottleneck configuration in a circuit that does not allow to eliminate more than three gates easily. To overcome this difficulty, we use a circuit complexity measure that depends on the number of troubled gates. This, in turn, requires us to analyze carefully how many new troubled gates can be introduced by applying a normalization rule. At the same time, we show that a circuit computing an affine disperser cannot have too many troubled gates (otherwise one could find an affine subspace of not too large dimension that makes the circuit constant). This implies that the bottleneck case cannot appear too often during the gate elimination process.

In Section III-C we formally define a source arising from constant, affine, and quadratic substitutions. We apply quadratic substitutions very carefully. In particular, we maintain the following invariant: the variables from the right-hand side of quadratic substitutions are pairwise different and do not appear in the left hand side of affine substitutions. This invariant guarantees that a disperser for affine sources is also a disperser for our generalized sources (with parameters that are only slightly worse).

In Section III-D we define the circuit complexity measure and formulate the main result: we can always reduce the measure by an appropriate amount by shrinking the space; the lower bound follows. The measure is defined as a linear combination of four parameters of a circuit: the number of gates, the number of troubled gates, the number of quadratic substitutions, and the number of inputs. The optimal values for coefficients in this linear combination come from solving a simple linear program.

Finally, Section III-E employs all developed techniques in order to prove the main lower bound of the paper. Due to the page limit, here we only present a proof sketch, the detailed proof can be found in the full version of the paper [58].

B. Cyclic circuit transformations

In this section we consider several types of substitutions. It is straightforward how to substitute a constant to an input:

Proposition 1. *Let C be a circuit with input gates $x_1, \ldots, x_n$, and let $c \in \{0,1\}$ be a constant. For every gate G fed by x_1 replace the operation $g(x_1, t)$ computed by G with the operation $g'(x_1, t) = g(c, t)$ (thus the result becomes independent of x_1). This transforms C into another circuit C' (in particular, it is still a fair semicircuit) such that it has the same number of gates, the same topology, and for every gate H that computes a function $h(x_1, \ldots, x_n)$ in C, the corresponding gate in the new circuit C' computes the function $h(c, x_2, \ldots, x_n)$.*

We call this transformation a *substitution by a constant*.

A more complicated type of a substitution is when we replace an input x with a function computed in a different gate G. In this case in each gate fed by x, we replace wires going from x by wires going from G. We call this transformation a *substitution by a function*.

Proposition 2. *Let C be a circuit with input gates $x_1, \ldots, x_n$, and let $g(x_2, \ldots, x_n)$ be a function computed* *in a gate G. Consider the construction C' obtained by substituting a function g to x_1 (it has the same number of gates as C). Then if G is not reachable from x_1 by a directed path in C, then C' is a fair semicircuit, and for every gate H that computes a function $h(x_1, \ldots, x_n)$ in C, except for x_1, the corresponding gate in the new circuit C' computes the function $h(g(x_2, \ldots, x_n), x_2, \ldots, x_n)$.*

In what follows, however, we will also use substitutions that do not satisfy the hypothesis of this proposition: substitutions that create cycles. We defer this construction to Section III-B2.

1) Normalization and troubled gates: In order to work with a circuit, we are going to assume that it does not contain obvious inefficiencies (such as trivial gates, etc.), in particular, those created by substitutions. We describe certain normalization rules below; however, while normalizing we need to make sure the circuit remains within certain limits: in particular, it must remain fair and compute the same function. We need to check also that we do not "spoil" a circuit by introducing "bottleneck" cases. Namely, we are going to prove an upper bound on the number of newly introduced unwanted fragments called "troubled" gates.

We say that an internal gate G is *troubled* if it satisfies the following three criteria: G is an and-type gate with outdegree 1, the gates feeding G are input gates, and both input gates feeding G have outdegree 2. *From now on, we denote all and-type gates by $\wedge$, and all xor-type gates by $\oplus$.*

We always make substitutions consciously and thus can count the number of troubled gates that can possible emerge. However, what if a gate is killed because of simplifications? We limit the process of removing gates to normalization rules, and make sure that we never get more than four new troubled gates per killed gate. We say that a circuit is *normalized* if none of the following rules is applicable to it. Each rule eliminates a gate G whose inputs are gates I_1 and I_2. (Note that I_1 and I_2 can be inputs or internal nodes, and, in rare cases, they can coincide with G itself).

Rule 1: If G has no outgoing edges and is not marked as an output, then remove it. Note also that it could not happen that the only outgoing edge of G feeds itself, because this would make a trivial equation and violate the circuit fairness.
Rule 2: If G is trivial, i.e., it computes a constant function c of the circuit inputs (not necessarily a constant operation on the two inputs of G), remove G and "embed" this constant to the next gates. That is, for every gate H fed by G, replace the operation $h(g, t)$ computed in this gate (where g is the input from G and t is the other input) by the operation $h'(g, t) = h(c, t)$. (Clearly, h' depends on at most one argument, which is not optimal, and in this case after removing G one typically applies Rule 3 or Rule 2 to its successors.)
Rule 3: If G is passing, i.e., it computes an operation depending only on one of its inputs, remove G by reattaching its outgoing wires to that input. This may also require changing the operations computed at its successors (the corresponding input may be negated; note that an and-type gate (xor-type gate) remains an and-type gate (xor-type gate)).

If G feeds itself and depends on another input, then the self-loop wire (which would now go nowhere) is dropped. (Note that if G feeds itself it cannot depend on the self-loop input.)

If G has no outgoing edges it must be an output gate (otherwise it would be removed by Rule 1). In this special case, we remove G and mark the corresponding input of G (or its negation) as the output gate.
Rule 4: If G is a 1-gate that feeds a single gate Q, Q is distinct from G itself, and Q is also fed by one of G's inputs, then replace in Q the incoming wire going from G by a wire going from the other input of G (this might also require changing the operation at Q); then remove G. We call such a gate G *useless*.
Rule 5: If the inputs of G coincide (I_1 and I_2 refer to the same node) then we replace the binary operation $g(x, y)$ computed in G with the operation $g'(x, y) = g(x, x)$. Then perform the same operation on G as described in Rule 3 or 2.

Proposition 3. *Each of the Rules 1–5 removes one internal gate, introduces at most four new troubled gates. An input gate that was not connected by a directed path to the output gate cannot be connected by a new directed path*[1]*. None of the rules change the functions of n input variables computed in the gates that are not removed. A fair semicircuit remains a fair semicircuit.*

2) Affine substitutions: In this section, we show how to make substitutions that do create cycles. This will be needed in order to make affine substitutions. Namely, we take a gate computing an affine function $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c$ (where $c \in \{0, 1\}$ is a constant) and "rewire" a circuit so that this gate is replaced by a trivial gate computing a constant $b \in \{0, 1\}$, while x_1 is replaced by an internal gate. The resulting circuit over $x_2, \ldots, x_n$ may be viewed as the initial circuit under the substitution $x_1 \leftarrow \bigoplus_{i \in I} x_i \oplus c \oplus b$. The "rewiring" is formally explained below; however, before that we need to prove a structural lemma (which is trivial for acyclic circuits) that guarantees its success.

For an xor-circuit, we say that a gate G depends on a variable x if G computes an affine function in which x is a term. Note that in a circuit without cycles this means that precisely one of the inputs of G depends on x, and one could trace this dependency all the way to x, therefore there always exists a path from x to G. The following lemma states that it is always possible to find such a path in a fair

[1]This trivial observation will be formally needed when we later count the number of such gates.

cyclic circuit too. However, it may be possible that some nodes on this path do not depend on x.

Lemma 1. *Let C be a fair cyclic xor-circuit, and let the gate G depend on the variable x. Then there is a path from x to G.*

We now come to rewiring.

Lemma 2. *Let C be a fair semicircuit with input gates $x_1, \ldots, x_n$ and internal gates $G_1, \ldots, G_m$. Let G be a gate not reachable by a directed path from any and-type gate. Assume that G computes the function $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c$, where $I \subseteq \{2, \ldots, n\}$. Let $b \in \{0,1\}$ be a constant. Then one can transform C into a new circuit C' with the following properties: 1) graph-theoretically, C' has the same gates as C, plus a new internal gate Z; some edges are changed, in particular, x_1 is disconnected from the circuit; 2) the operation in G is replaced by the constant operation b; 3) $\mathrm{in}_{C'}(Z) = 2$, $\mathrm{out}_{C'}(G) = \mathrm{out}_C(G) + 1$, $\mathrm{out}_{C'}(x_1) = 0$. $\mathrm{out}_{C'}(Z) = \mathrm{out}_C(x_1) - 1$. 4) The indegrees and outdegrees of all other gates are the same in C and C'. 5) C' is fair. 6) all gates common for C' and C compute the same functions on the affine subspace defined by $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c \oplus b = 0$, that is, if $f(x_1, \ldots, x_n)$ is the function computed by an internal gate in C and $f'(x_2, \ldots, x_n)$ is the function computed by its counterpart in C', then $f(\bigoplus_{i \in I} x_i \oplus c \oplus b, x_2, \ldots, x_n) = f'(x_2, \ldots, x_n)$. The gate Z computes the function $\bigoplus_{i \in I} x_i \oplus c \oplus b$ (which on the affine subspace equals x_1).*

This transformation does not introduce new troubled gates.

After we apply the transformation, we apply Rule 2 to G. Since the only troubled gates introduced by this rule are the inputs of the removed gate, no troubled gates are introduced (and one gate, G itself, is eliminated, thus the combination of Lemma 2 and Rule 2 does not increase the number of gates).

C. Read-once depth-2 quadratic sources

We generalize affine sources as follows.

Definition 1. *Let the set of variables $\{x_1, \ldots, x_n\}$ be partitioned into three disjoint sets $F, L, Q \subseteq \{1, \ldots, n\}$ (for* free*,* linear*, and* quadratic*). Consider a system of equalities that contains for each variable x_j with $j \in Q$, a quadratic equality of the form $x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j$, where $i, k \in F$ and c_i, c_k, c_j are constants; the variables from the right-hand side of all the quadratic substitutions are pairwise disjoint. For each variable x_j with $j \in L$, an affine equality of the form $x_j = \bigoplus_{i \in F_j \subseteq F} x_i \oplus \bigoplus_{i \in Q_j \subseteq Q} x_i \oplus c_j$ for some constant c_j. A subset R of $\{(x_1, x_2, \ldots, x_n) \in \mathbb{F}_2^n\}$ that satisfies these equalities is called a* read-once depth-2 quadratic source *(or* rdq-source*) of dimension $d = |F|$.*

The variables from the right-hand side of quadratic substitutions are called *protected*. Other free variables are called *unprotected*.

For this, we will gradually build a straight-line program (that is, a sequence of lines of the form $x = f(\ldots)$, where f is a function depending on the program inputs (free variables) and the values computed in the previous lines) that produces an rdq-source. We build it bottom-up. Namely, we take an unprotected free variable x_j and extend our current program with either a quadratic substitution $x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j$ depending on free unprotected variables x_i, x_k or a linear substitution $x_j = \bigoplus_{i \in J} x_i \oplus c_j$ depending on any variables. It is clear that such a program can be rewritten into a system satisfying Definition 1. In general, we cannot use protected free variables without breaking the rdq-property. However, there are two special cases where this is possible: (1) we can substitute a constant to a protected variable (and update the quadratic line accordingly: for example, $z = xy$ and $x = 1$ yield $z = y$ and $x = 1$); (2) we can substitute one protected variable for another variable (or its negation) from the same quadratic equation (for example, $z = xy$ and $x = y$ yield $z = y$ and $x = y$).

In what follows we abuse the notation by denoting by the same letter R the source, the straight-line program defining it, and the mapping $R \colon \mathbb{F}_2^d \to \mathbb{F}_2^n$ computed by this program that takes the d free variables and evaluates all other variables.

Let $R \subseteq \mathbb{F}_2^n$ be an rdq-source of dimension d, let the free variables be $x_1, x_2, \ldots, x_d$, and let $f \colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a function. Then f restricted to R, denoted $f|_R$, is a function $f|_R \colon \mathbb{F}_2^d \to \mathbb{F}_2$, defined by $f|_R(x_1, \ldots, x_d) = f(R(x_1, \ldots, x_d))$. Note that affine sources are precisely rdq-sources with $Q = \emptyset$. We define dispersers for rdq-sources similarly to dispersers for affine sources: A family of functions $f_n \colon \mathbb{F}_2^n \to \mathbb{F}_2$ is an *rdq-disperser* for dimension $d(n)$ if for all sufficiently large n, for every rdq-source R of dimension at least $d(n)$, $f_n|_R$ is non-constant. The following proposition shows that affine dispersers are also rdq-dispersers for related parameters. By setting one protected variable to 0 for each quadratic restriction, we get that if R is an rdq-source of $\mathbb{F}_2^n$ of dimension d, then R contains an affine subspace of dimension at least $d/2$.[2] In particular we have the following.

Corollary 1. *An affine disperser for dimension d is an rdq-disperser for dimension $2d$. In particular, an affine disperser for sublinear dimension is also an rdq-disperser for sublinear dimension.*

[2]This is obviously false for quadratic *varieties*: no Boolean function can be non-constant on all sets of common roots of $n - o(n)$ quadratic polynomials. For example, the system of $n/2$ quadratic equations $x_1x_2 = x_3x_4 = \ldots = x_{n-1}x_n = 1$ defines a single point, so any function is constant on this set.

D. Circuit complexity measure

For a circuit C and a straight-line program R defining an rdq-source (over the same set of variables), we define the following circuit complexity measure:

$$\mu(C, R) = g + \alpha_Q \cdot q + \alpha_T \cdot t + \alpha_I \cdot i\,,$$

where g is the number of internal gates in C, q is the number of quadratic substitutions in R, t is the number of troubled gates in C, and i is the number of *influential* input gates in C. We say that an input is influential if it feeds at least one gate or is protected (recall that a variable is protected if it occurs in the right-hand side of a quadratic substitution in R). The constants $\alpha_Q, \alpha_T, \alpha_I > 0$ will be chosen later.

Proposition 3 implies that when a gate is removed from a circuit by applying a normalization rule the measure μ is reduced by at least $\beta = 1 - 4\alpha_T$. The constant α_T will be chosen to be very close to 0 (certainly less than $1/4$), so $\beta > 0$.

In order to estimate the initial value of our measure, we need the following lemma.

Lemma 3. *Let C be a circuit computing an affine disperser $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ for dimension d, then the number of troubled gates in C is less than $\frac{n}{2} + \frac{5d}{2}$.*

We are now ready to formulate our main result.

Theorem 1. *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be an rdq-disperser for dimension d and C be a fair semicircuit computing f. Let $\alpha_Q, \alpha_T, \alpha_I \geq 0$ be some constants, and $\alpha_T \leq 1/4$. Then $\mu(C, \emptyset) \geq \delta(n - d - 2)$ where*

$$\delta := \alpha_I + \min\left\{\frac{\alpha_I}{2}, 4\beta, 3 + \alpha_T, 2\beta + \alpha_Q, 5\beta - \alpha_Q, 2.5\beta + \frac{\alpha_Q}{2}\right\}\,,$$

and $\beta = 1 - 4\alpha_T$.

We defer the proof of this theorem to the next section. This theorem, together with Corollary 1 and Lemma 3, implies a lower bound on the circuit complexity of affine dispersers.

Corollary 2. *Let $\delta, \beta, \alpha_Q, \alpha_T, \alpha_I$ be constants as above, then the circuit size of an affine disperser for sublinear dimension is at least $\left(\delta - \frac{\alpha_T}{2} - \alpha_I\right) n - o(n)$.*

The maximal value of $\delta - \frac{\alpha_T}{2} - \alpha_I$ satisfying the condition from Corollary 2 is achieved when $\alpha_T = \frac{1}{43}$, $\alpha_Q = \frac{65}{43}$, $\alpha_I = 6 + \frac{2}{43}$, $\beta = \frac{39}{43}$, $\delta = 9 + \frac{3}{43}$. This gives the main result of the paper.

Main Theorem. *The circuit size of an affine disperser for sublinear dimension is at least $(3 + \frac{1}{86})n - o(n)$.*

E. Gate elimination

In order to prove Theorem 1 we first show that it is always possible to make a substitution and decrease the measure by δ. The main theorem then follows by a simple induction proof.

Theorem 2. *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be an rdq-disperser for dimension d, let R be an rdq-source of dimension $s \geq d + 2$, and let C be an optimal (i.e., C with the smallest $\mu(C, R)$) fair semicircuit computing the function $f|_R$. Then there exist an rdq-source R' of dimension $s' < s$ and a fair semicircuit C' computing the function $f|_{R'}$ such that $\mu(C', R') \leq \mu(C, R) - \delta(s - s')$.*

The proof of Theorem 2 is based on a careful consideration of a number of cases. Due to the page limit restrictions, here we show a high-level picture of the case analysis only.

We fix the values of constants $\alpha_T, \alpha_Q, \alpha_I, \beta, \delta$ to the optimal values: $\alpha_T = \frac{1}{43}, \alpha_Q = \frac{65}{43}, \alpha_I = 6\frac{2}{43}, \beta = \frac{39}{43}, \delta = 9\frac{3}{43}$. Now it suffices to show that we can always make one substitution and decrease the measure by at least $\delta = 9\frac{3}{43}$. First we normalize the circuit. By Proposition 3, during normalization if we eliminate a gate then we introduce at most four new troubled gates, this means that we decrease the measure by at least $1 - 4\alpha_T = \frac{39}{43}$. Therefore, normalization never increases the measure.

We always make constant, linear or simple quadratic *substitution* to a variable. Then we remove the substituted variable from the circuit, so that for each assignment to the remaining variables the function is defined. It is easy to make a constant substitution $x = c$ for $c \in \{0, 1\}$. We propagate the value c to the inputs fed by x and remove x from the circuit, since it does not feed any other gates. An affine substitution $x = \bigoplus_{i \in I} x_i \oplus c$ is harder to make, because a straightforward way to eliminate x would be to compute $\left(\bigoplus_{i \in I} x_i \oplus c\right)$ elsewhere. We will always have a gate G that computes $\bigoplus_{i \in I} x_i \oplus c$ and that is not reachable by a direct path from an and-type gate. Fortunately, in this case Lemma 2 shows how to compute it on the affine subspace defined by the substitution without using x and without increasing the number of gates (later, an extra gate introduced by this lemma is removed by normalization). Thus, in this sketch we will be making arbitrary affine substitutions for sums that are computed in gates without saying that we need to run the reconstruction procedure first. Also, we will make a simple quadratic substitution $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$ only if the gates fed by z are cancelled out after the substitution, so that we do not need to propagate this quadratic value to other gates. We want to stay in the class of rdq-sources, therefore we cannot make an affine substitution to a variable x if it already has been used in the right-hand side of some quadratic restriction $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$ (that is, x is protected), also we cannot make quadratic substitutions that overlap in the

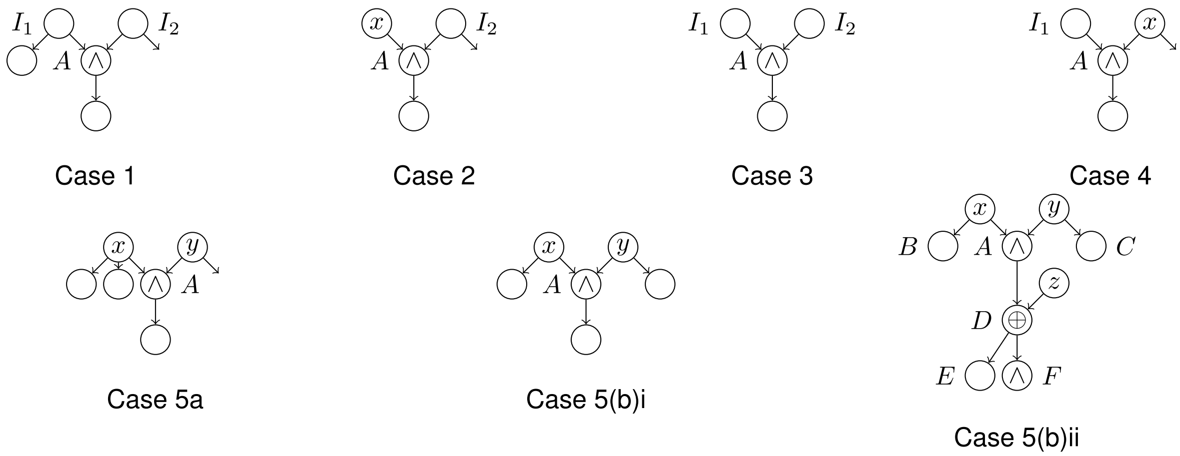

Figure 1: Gate elimination process in Theorem 2.

variables. In this proof sketch we ignore these two issues, but they are addressed in the full version of the paper.

Let A be a topologically minimal and-type gate (i.e., an and-type gate that is not reachable from any and-type gate), let I_1 and I_2 be the inputs of A (I_1 and I_2 can be variables or internal gates). Now we consider the following cases (see Figure 1).

1) At least one of I_1, I_2 (say, I_1) is an internal gate of outdegree greater than one. There is a constant c such that if we assign $I_1 = c$, then A becomes constant. (For example, if A is an and, then $c = 0$, if A is an or, then $c = 1$ etc.) When A becomes constant it eliminates all the gates it feeds. Therefore, if we assign the appropriate constant to I_1, we eliminate I_1, two of the gates it feeds (including A), and also a successor of A, four gates total, and we decrease the measure by at least $\alpha_I + 4\beta = 9\frac{29}{43} > \delta$.
2) At least one of I_1, I_2 (say, I_1) is a variable of outdegree one. We assign the appropriate constant to I_2. This eliminates I_2, A, a successor of A, and I_1. This assignment eliminates at least two gates and two variables, so the measure decrease is at least $2\alpha_I + 2\beta = 13\frac{39}{43} > \delta$.
3) I_1 and I_2 are internal gates of outdegree one. Then if we assign the appropriate constant to I_1, we eliminate I_1, A, a successor of A, and I_2 (since I_2 does not feed any gates). We decrease measure by at least $\alpha_I + 4\beta > \delta$.
4) I_1 is an internal gate of outdegree one, I_2 is a variable of outdegree greater than one. Then we assign the appropriate constant to I_2. This assignment eliminates I_2, at least two of its successors (including A), a successor of A, and I_1 (since it does not feed any gates). Again, we decrease the measure by at least $\alpha_I + 4\beta > \delta$.
5) I_1 and I_2 are variables of outdegree greater than one.
   a) I_1 or I_2 (say, I_1) has outdegree at least three. By assigning the appropriate constant to I_1 we eliminate at least three of the gates it feeds and a successor of A, four gates total.
   b) I_1 and I_2 are variables of degree two. If A is a 2^+-gate we eliminate at least four gates by assigning I_1 so in what follows we assume that A is a 1-gate. In this case A is a troubled gate. We want to make the appropriate substitution and eliminate I_1 (or I_2), its successor, A, and A's successor.
      i) If this substitution does not introduce new troubled gates, then we eliminate a variable, three gates and decrease the number of troubled gates by one. Thus, we decrease the measure by $\alpha_I + 3 + \alpha_T = 9\frac{3}{43} = \delta$.
      ii) If the substitution introduces troubled gates, then we consider which normalization rule introduces troubled gates. The full case analysis is presented in the full paper, here we demonstrate just one case of the analysis. Let us consider the case when a new troubled gate is introduced when we eliminate the gate fed by A (see Figure 1, the variable z will feed a new troubled gate after assignments $x = 0$ or $y = 0$). In such a case we make a different substitution: $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$. This substitution eliminates gates A, D, E, F and a gate fed by F. Thus, we eliminate one variable, five gates, but we introduce a new quadratic substitution, and decrease the measure by at least $\alpha_I + 5\beta - \alpha_Q = 9\frac{3}{43} = \delta$.

It is conceivable that when we count several eliminated gates, some of them coincide, so that we actually eliminate fewer gates. Usually in such cases we can prove that some other gates become trivial. This and other degenerate cases are handled in the full version of the paper [58].

Acknowledgements

The research presented in Sections III.B, III.D, and III.E is supported by Russian Science Foundation (project 16-11-10123). The research presented in Section III.C is supported by NSF grant 1319051.

We would like to thank Dmitry Itsykson and Alexander Knop for their valuable comments on earlier versions, and Olga Melanich for proofreading the manuscript. We also would like to thank the anonymous reviewers for their helpful comments.

References

[1] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell Systems Technical Journal*, vol. 28, pp. 59–98, 1949.

[2] H. Buhrman, L. Fortnow, and T. Thierauf, "Nonrelativizing separations," in *CCC-98*, 1998.

[3] V. T. Chakaravarthy and S. Roy, "Oblivious symmetric alternation," in *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science*, 2006, pp. 230–241.

[4] B. M. Kloss and V. A. Malyshev, "Estimates of the complexity of certain classes of functions," *Vestn.Moskov.Univ.Ser.1*, vol. 4, pp. 44–51, 1965, in Russian.

[5] C. Schnorr, "Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen," *Computing*, vol. 13, no. 2, pp. 155–171, 1974.

[6] L. J. Stockmeyer, "On the combinational complexity of certain symmetric Boolean functions," *Mathematical Systems Theory*, vol. 10, pp. 323–336, 1977.

[7] W. J. Paul, "A $2.5n$-lower bound on the combinational complexity of Boolean functions," *SIAM J. Comput.*, vol. 6, no. 3, pp. 427–443, 1977.

[8] N. Blum, "A Boolean function requiring $3n$ network size," *Theor. Comput. Sci.*, vol. 28, pp. 337–345, 1984.

[9] E. Demenkov and A. S. Kulikov, "An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers," in *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS*, 2011, pp. 256–265.

[10] S. Vadhan and R. Williams, "Personal communication," 2013.

[11] C. Schnorr, "The combinational complexity of equivalence," *Theor. Comput. Sci.*, vol. 1, no. 4, pp. 289–295, 1976.

[12] U. Zwick, "A $4n$ lower bound on the combinational complexity of certain symmetric Boolean functions over the basis of unate dyadic Boolean functions," *SIAM J. Comput.*, vol. 20, no. 3, pp. 499–505, 1991.

[13] O. Lachish and R. Raz, "Explicit lower bound of $4.5n - o(n)$ for Boolean circuits," in *Proceedings on 33rd Annual ACM Symposium on Theory of Computing*, J. S. Vitter, P. G. Spirakis, and M. Yannakakis, Eds. ACM, 2001, pp. 399–408.

[14] K. Iwama and H. Morizumi, "An explicit lower bound of $5n-o(n)$ for Boolean circuits," in *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS*, 2002, pp. 353–364.

[15] E. Demenkov, A. S. Kulikov, O. Melanich, and I. Mihajlin, "New lower bounds on circuit size of multi-output functions," *Theory Comput. Syst.*, vol. 56, no. 4, pp. 630–642, 2015.

[16] K. Amano and J. Tarui, "A well-mixed function with circuit complexity 5n: Tightness of the Lachish-Raz-type bounds," *Theor. Comput. Sci.*, vol. 412, no. 18, pp. 1646–1651, 2011.

[17] G. Cohen and A. Tal, "Two structural results for low degree polynomials and applications," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, 2015, pp. 680–709.

[18] A. A. Razborov, "Lower bound on monotone complexity of some Boolean functions," *Doklady Akademii Nauk. SSSR*, vol. 281, no. 4, pp. 798–801, 1985.

[19] A. C. Yao, "Separating the polynomial-time hierarchy by oracles (preliminary version)," in *FOCS*. IEEE Computer Society, 1985, pp. 1–10.

[20] J. Håstad, "Almost optimal lower bounds for small depth circuits," in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, 1986, pp. 6–20.

[21] V. Shoup and R. Smolensky, "Lower bounds for polynomial evaluation and interpolation problems," in *32nd Annual Symposium on Foundations of Computer Science*, 1991, pp. 378–383.

[22] B. A. Subbotovskaya, "Realizations of linear functions by formulas using $+, \cdot, -$," *Doklady Akademii Nauk. SSSR*, vol. 136, no. 3, pp. 553–555, 1961.

[23] V. M. Khrapchenko, "A method of determining lower bounds for the complexity of π-schemes," *Math. Notes of the Acad. of Sci. of the USSR*, vol. 10, no. 1, pp. 474–479, 1971.

[24] E. I. Nechiporuk, "On a Boolean function," *Doklady Akademii Nauk. SSSR*, vol. 169, no. 4, pp. 765–766, 1966.

[25] A. E. Andreev, "On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes," *Moscow Univ. Math. Bull.*, vol. 42, no. 1, pp. 63–66, 1987.

[26] R. Impagliazzo and N. Nisan, "The effect of random restrictions on formula size," *Random Struct. Algorithms*, vol. 4, no. 2, pp. 121–134, 1993.

[27] M. Paterson and U. Zwick, "Shrinkage of de Morgan formulae under restriction," *Random Struct. Algorithms*, vol. 4, no. 2, pp. 135–150, 1993.

[28] J. Håstad, "The shrinkage exponent of de Morgan formulas is 2," *SIAM J. Comput.*, vol. 27, no. 1, pp. 48–64, 1998.

[29] A. Tal, "Shrinkage of de Morgan formulae by spectral techniques," in *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 2014, pp. 551–560.

[30] R. Williams, “Improving exhaustive search implies superpolynomial lower bounds,” *SIAM J. Comput.*, vol. 42, no. 3, pp. 1218–1244, 2013, extended abstract appeared in Proc. STOC-2010.

[31] ——, “Nonuniform ACC circuit lower bounds,” *JACM*, vol. 61, no. 1, 2014, extended abstract appears in Proc. CCC-2011.

[32] E. Ben-Sasson and E. Viola, “Short PCPs with projection queries,” in *Automata, Languages, and Programming - 41st International Colloquium*, 2014, pp. 163–173.

[33] S. Nurk, “An $2^{0.4058m}$ upper bound for Circuit SAT,” Steklov Institute of Mathematics at St.Petersburg, Tech. Rep. 10, 2009, PDMI Preprint.

[34] S. Savinov, “Upper bounds for the Boolean circuit satisfiability problem,” Master Thesis defended at St.Peterburg Academic University of Russian Academy of Sciences, 2014, in Russian.

[35] R. Santhanam, “Fighting perebor: New and improved algorithms for formula and QBF satisfiability,” in *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2010, pp. 183–192.

[36] K. Seto and S. Tamaki, “A satisfiability algorithm and average-case hardness for formulas over the full binary basis,” *Computational Complexity*, vol. 22, no. 2, pp. 245–274, 2013.

[37] I. Komargodski, R. Raz, and A. Tal, “Improved average-case lower bounds for demorgan formula size,” in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2013, pp. 588–597.

[38] R. Chen, V. Kabanets, A. Kolokolova, R. Shaltiel, and D. Zuckerman, “Mining circuit lower bound proofs for meta-algorithms,” *Computational Complexity*, vol. 24, no. 2, pp. 333–392, 2015.

[39] R. Chen and V. Kabanets, “Correlation bounds and #sat algorithms for small linear-size circuits,” in *Computing and Combinatorics - 21st International Conference, COCOON*, 2015, pp. 211–222.

[40] E. Ben-Sasson and S. Kopparty, “Affine dispersers from subspace polynomials,” *SIAM J. Comput.*, vol. 41, no. 4, pp. 880–914, 2012.

[41] R. L. Rivest, “The necessity of feedback in minimal monotone combinational circuits,” *IEEE Trans. Computers*, vol. 26, no. 6, pp. 606–607, 1977.

[42] P. W. Dymond and S. A. Cook, “Complexity theory of parallel time and hardware,” *Inf. Comput.*, vol. 80, no. 3, pp. 205–226, 1989.

[43] A. Nickelsen, T. Tantau, and L. Weizsäcker, “Aggregates with component size one characterize polynomial space,” *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 028, 2004.

[44] M. D. Riedel and J. Bruck, “Cyclic boolean circuits,” *Discrete Applied Mathematics*, vol. 160, no. 13-14, pp. 1877–1900, 2012.

[45] A. Kojevnikov and A. S. Kulikov, “Circuit complexity and multiplicative complexity of Boolean functions,” in *Programs, Proofs, Processes, 6th Conference on Computability in Europe, CiE*, 2010, pp. 239–245.

[46] O. Kullmann, “New methods for 3-SAT decision and worst-case analysis,” *Theor. Comput. Sci.*, vol. 223, no. 1-2, pp. 1–72, 1999.

[47] A. Kojevnikov and A. S. Kulikov, “A new approach to proving upper bounds for MAX-2-SAT,” in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2006, pp. 11–17.

[48] F. V. Fomin, F. Grandoni, and D. Kratsch, “A measure & conquer approach for the analysis of exact algorithms,” *J. ACM*, vol. 56, no. 5, 2009.

[49] A. Golovnev and A. S. Kulikov, “Weighted gate elimination: Boolean dispersers for quadratic varieties imply improved circuit lower bounds,” in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, 2016, pp. 405–411.

[50] Z. Dvir, “Extractors for varieties,” *Computational Complexity*, vol. 21, no. 4, pp. 515–572, 2012.

[51] R. Shaltiel, “Dispersers for affine sources with sub-polynomial entropy,” in *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, 2011, pp. 247–256.

[52] A. Golovnev, A. S. Kulikov, A. Smal, and S. Tamaki, “Circuit size lower bounds and #SAT upper bounds through a general framework,” *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 23, 2016.

[53] B. Barak, G. Kindler, R. Shaltiel, B. Sudakov, and A. Wigderson, “Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors,” *J. ACM*, vol. 57, no. 4, 2010.

[54] J. Bourgain, “On the construction of affine extractors,” *GAFA Geometric And Functional Analysis*, vol. 17, no. 1, pp. 33–57, 2007.

[55] A. Yehudayoff, “Affine extractors over prime fields,” *Combinatorica*, vol. 31, no. 2, pp. 245–256, 2011.

[56] X. Li, “A new approach to affine extractors and dispersers,” in *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC*, 2011, pp. 137–147.

[57] ——, “Extractors for affine sources with polylogarithmic entropy,” *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 22, 2015.

[58] M. G. Find, A. Golovnev, E. A. Hirsch, and A. S. Kulikov, “A better-than-$3n$ lower bound for the circuit complexity of an explicit function,” *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 22, 2015, revision 1.