Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 166 (2015)

Improving $3n$ Circuit Complexity Lower Bounds

Magnus Gausdal Find   Alexander Golovnev*   Edward A. Hirsch†
Alexander S. Kulikov‡

August 16, 2022

Abstract

Proving circuit lower bounds remains a tremendously difficult problem. Although it can be easily shown by counting that almost all Boolean predicates of n variables have circuit size $\Omega(2^n/n)$, we have no example of an **NP** function requiring even a superlinear number of gates. Moreover, only modest linear lower bounds are known. Until recently, the strongest known lower bound was $3n - o(n)$ presented by Blum in 1984. In 2011, Demenkov and Kulikov presented a much simpler proof of the same lower bound, but for a more complicated function — an affine disperser for sublinear dimension. Informally, this is a function that is resistant to any $n - o(n)$ affine substitutions. In 2011, Ben-Sasson and Kopparty gave an explicit construction of such a function. The proof of the lower bound basically goes by showing that for any circuit there exists an affine hyperplane where the function complexity decreases at least by three gates. In this paper, we prove the following two extensions.

1. We prove a $\left(3 + \frac{1}{86}\right) n - o(n)$ lower bound for the circuit size of an affine disperser for sublinear dimension. 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.
2. We then present a much simpler proof of a stronger lower bound of $3.11n$ for a quadratic disperser. Informally, a quadratic disperser is resistant to sufficiently many substitutions of the form $x \leftarrow p$, where p is a polynomial of degree at most two. Currently, there are no constructions of quadratic dispersers in **NP** (although there are constructions over large fields, and constructions with weaker parameters over GF(2)). The key ingredient of this proof is the induction on the size of the underlying quadratic variety instead of the number of variables as in the previously known proofs.

1 Introduction

1.1 Overview

Denote by $B_{n,m}$ the set of all Boolean functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$, let $B_n = B_{n,1}$ and consider a function $f \in B_n$. A natural question studied in theoretical computer science is the following: what is the minimal number of binary Boolean operations needed to compute f? The corresponding computational model is Boolean circuits. A circuit is a directed acyclic graph with inputs $x_1, \ldots, x_n$, the intermediate vertices have indegree 2 and are labeled with binary Boolean operations. The size of a circuit is its number of internal gates (these intermediate vertices). Note that we do not impose any restrictions on the depth or outdegree. By $C(f)$ we denote the minimum size of a circuit computing f.

*Georgetown University.
†Technion.
‡Steklov Institute of Mathematics at St. Petersburg, Russian Academy of Sciences; St.Petersburg State University.

ISSN 1433-8092

Counting shows that the number of small size circuits is much smaller than the total number $|B_n| = 2^{2^n}$ of functions. Using this idea it was shown by Shannon [55] that almost all functions from B_n require circuits of size $\Omega(2^n/n)$. This proof is however non-constructive: it does not give an explicit function of high circuit complexity. Showing superpolynomial lower bounds for explicitly defined functions (for example, for a function in **NP**) remains an incredibly challenging task. In particular, such lower bounds would imply $\mathbf{P} \neq \mathbf{NP}$. Moreover, superlinear bounds are unknown even for functions in $\mathbf{E}^{\mathbf{NP}}$. Superpolynomial bounds are known for **MAEXP** (exponential-time Merlin-Arthur games) [7] and $\mathbf{ZPEXP}^{\mathbf{MCSP}}$ (exponential-time **ZPP** with oracle access to the Minimal Circuit Size Problem) [24], and arbitrary polynomial lower bounds are known for $\mathbf{S_2}$ (the symmetric second level of the polynomial hierarchy) which is the smallest known class both containing **NP** and to which **PH** collapses by a Karp-Lipton-style theorem [8]. Also arbitrary polynomial lower bounds are known for a slightly non-uniform class **MA/1** via padding a **PSPACE**-hard language and employing a Karp-Lipton-style theorem for **PSPACE** [47].

People started to tackle the problem in the 60s. Kloss and Malyshev [28] proved a lower bound of $2n - O(1)$. Schnorr [50] proved a $2n - O(1)$ lower bound for a class of functions with certain structure. Stockmeyer [57] proved a $2.5n - O(1)$ bound for certain symmetric functions. Paul [43] proved a $2.5n - o(n)$ lower bound for a variant of the storage access function. Eventually, Blum [6] extended Paul's argument and proved a $3n - o(n)$ bound.

Blum's bound remained unbeaten for more than thirty years. Blum's proof consists of two parts: one part constitutes a gate elimination procedure similar in spirit to previous bounds, the other part considers the topology of paths in the circuit and it is designing specifically for proving a lower bound of $3n - o(n)$.

Demenkov and Kulikov [14] presented a much simpler proof of a $3n-o(n)$ lower bound for functions with an entirely different property: affine dispersers. This property allows one to restrict the function to smaller and smaller affine subspaces until the disperser's dimension is reached. As was later noted by Vadhan and Williams [60], the way Demenkov and Kulikov use this property cannot give stronger than $3n$ bounds as the technique from [14] gives a tight bound of $n - 1$ for the inner product function (which is known to be an affine disperser for dimension $n/2 + 1$).

Hence two different proofs using two different properties are both stuck at 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 improve the bound for affine dispersers to $(3 + \frac{1}{86})n - o(n)$, which is stronger than Blum's bound. We then show that a stronger lower bound of $3.11n$ can be proved much more easily for a stronger object that we call a quadratic disperser. Roughly, such a function is resistant to sufficiently many substitutions of the form $x \leftarrow p$ where p is a polynomial of degree at most 2. Currently, there are no constructions of quadratic dispersers in **NP** (there are constructions with weaker parameters for the field of size two and constructions for larger fields, though).

In a recent work, Li and Yang [26] extended the case analysis of the present paper (as well as the circuit complexity measure used in it) in a clever way to improve the lower bound on the complexity of affine dispersers to $3.1n - o(n)$.

1.2 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 (if NOT gates are not counted: the negations are written over the 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 (machines that work for all input lengths). 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

[51] proved that the circuit complexity of the parity function is $3n-3$. Zwick [66] gave a $4n-O(1)$ lower bound for certain symmetric functions, Lachish and Raz [35] 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 [25] 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 circuit lower bounds is also stuck at the $5n-o(n)$ lower bound: Amano and Tarui [1] presented an $(n-o(n))$-mixed function whose circuit complexity over U_2 is $5n+o(n)$.

While we do not have nonlinear bounds for constant-arity Boolean circuits, stronger bounds are known for weaker models, including monotone circuits (Razborov [44]), circuits of constant depth with no XOR gates (Yao [64] and Håstad [20]), circuits of polylogarithmic depth over infinite fields (Shoup and Smolensky [56]), formulas (Subbotovskaya [58], Khrapchenko [27], Nechiporuk [39], Andreev [2], Impagliazzo and Nisan [23], Paterson and Zwick [42], Håstad [21] and Tal [59]). These bounds, however, do not translate to superlinear lower bounds for general constant-arity Boolean circuits.

1.3 Connections to CircuitSAT algorithms

A recent promising direction initiated by Williams [62] 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 **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) [63]. Ben-Sasson and Viola [5] have demonstrated that for a constant k, one can prove a linear lower bound kn for a function in $\mathbf{E}^{\mathbf{NP}}$ by designing a 2^{cn}-time algorithm for 3-SAT for $c<1/(3k+1)$. It should be noted, however, that currently available 3-SAT algorithms are not sufficient for proving new lower bounds.

Also techniques similar to the ones used in proving circuit lower bounds are employed in a number of algorithms for CircuitSAT and FormulaSAT, see e.g. [41, 49, 48, 52, 32, 11, 10].

1.4 Affine Dispersers

In this paper, we prove lower bounds for affine (and quadratic) dispersers. Informally, an affine disperser is a function that cannot be made constant by sufficiently many linear substitutions. Formally, a function $f \in B_n$ is called an affine disperser for dimension d if it is not constant on any affine subspace of $\mathbb{F}_2^n$ of dimension at least d.

The notion of dispersers is a relaxation of the notion of extractors — functions that take input from a possibly non-uniform distribution and "extract" randomness from it: output a bit that is distributed statistically close to uniform. Unlike extractors, dispersers are only required to output a non-constant bit. To specify the class of input distributions, one defines a class of sources $\mathcal{F}$, where each $X \in \mathcal{F}$ is a distribution over $\mathbb{F}_2^n$. Since dispersers are only required to output a non-constant bit, we identify a distribution X with its support on $\mathbb{F}_2^n$. A function $f \in B_n$ is called a disperser for a class of sources $\mathcal{F}$, if $|f(X)|=2$ for every $X \in \mathcal{F}$. Since it is impossible to extract even one non-constant bit from an arbitrary source even if the source is guaranteed to have $n-1$ bits of entropy (each function from B_n is constant on 2^{n-1} inputs), many special cases of sources are studied (see [54] for an excellent survey). The sources we are focused on in this paper are affine sources and their generalization — sources for polynomial varieties. Affine dispersers have drawn much interest lately. In particular, Ben-Sasson and Kopparty [4] gave an explicit construction of affine dispersers for sublinear dimension $d=o(n)$, which was followed by other constructions of [65, 36, 53, 3, 37, 9]. Dispersers for polynomial varieties over large fields were studied by Dvir [16], and dispersers over $\mathbb{F}_2$ were studied by Cohen and Tal [13].

1.5 Gate elimination

Essentially, the only known technique for proving lower bounds for circuits with no restrictions on depth and outdegree is the gate elimination method. To illustrate it, let us give a proof of a $2n - O(1)$ lower bound presented by Schnorr [50]. The $\mathrm{MOD}^n_{3,r} \in B_n$ function outputs 1 if and only if the sum (over integers) of n input bits is congruent to r modulo 3. We prove that $\mathrm{MOD}^n_{3,r}$ requires circuits of size at least $2n - 6$ by induction on n. The base case $n \leq 3$ clearly holds. For the induction step consider an optimal circuit C computing $\mathrm{MOD}^n_{3,r}$ and a topologically minimal gate A in C (such a gate exists since for $n \geq 4$, $\mathrm{MOD}^n_{3,r}$ is not constant). Let x and y be input variables to A. The crucial observation is that either x or y must feed at least one other gate. Indeed if both x and y feed only A then the whole circuit depends on x and y only through A. This, in particular, means that by fixing x and y in four possible ways $((x, y) = (0, 0), (0, 1), (1, 0), (1, 1))$ one gets at most two different subfunctions while there must be three different subfunctions under these assignments: $\mathrm{MOD}^{n-2}_{3,0}$, $\mathrm{MOD}^{n-2}_{3,1}$, and $\mathrm{MOD}^{n-2}_{3,2}$ (they are pairwise different for $n \geq 4$). Assume that it is x that feeds at least one other gate and call it B. We then replace x by 0. This eliminates at least two gates from the circuit (A and B): if one of the inputs to a gate computes a constant, then this gate computes either a constant or a unary function on the other input and hence can be eliminated from the circuit. The resulting circuit computes the function $\mathrm{MOD}^{n-1}_{3,r}$ so the lower bound follows by induction. The best known lower bound for $\mathrm{MOD}^n_{3,r}$ is now $2.5n - O(1)$ by Stockmeyer [57], the best known upper bound is $3n - 5 - [(n + r) \bmod 3 = 0]$ by Kulikov and Slezkin [33]. Knuth [29] (see solution to exercise 480) conjectured that this upper bound is tight.

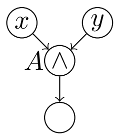
In the analysis above, we eliminated two gates by assigning $x \leftarrow 0$. If A computes, say, $xy = x \wedge y$ then we would have eliminated more than two gates since A becomes equal to 0 and hence all its successors are also eliminated. So, the bottleneck case is when both A and B compute parities of their inputs. In this case we cannot make A and B constant just by assigning a constant to x.

A natural idea that allows to overcome the bottleneck from the previous proof is to allow to substitute variables not only by constants but also by sums (over $\mathbb{F}_2$) of other variables. Using this idea one can prove a $3n - o(n)$ lower bound. The proof is due to Demenkov and Kulikov [14], the exposition here is due to Vadhan and Williams [60].

For a $3n - o(n)$ lower bound it is convenient to use xor-inputs circuits. In an xor-inputs circuit we allow linear sums of variables to be used as inputs to a circuit. Consider the following measure of an xor-inputs circuit C: $\mu(C) = G(C) + I(C)$ where $G(C)$ is the number of non-input gates and $I(C)$ is the number of inputs of C. Note that an xor-gate that depends on two inputs of an xor-inputs circuit C may be replaced by an input without increasing $\mu(C)$.

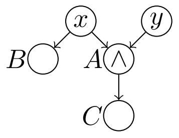
A $3n - 4d$ lower bound for the size of a (normal or xor-inputs) circuit computing an affine disperser $f \in B_n$ for dimension d follows from the following fact: for any affine subspace $S \subseteq \mathbb{F}_2^n$ of dimension D and any xor-inputs circuit C computing f on S, $\mu(C) \geq 4(D - d - 1)$ (now plug in $D = n$). The fact can be shown by induction on D. The base case $D \leq d + 1$ is clear. For the induction step, assume that C has the minimal value of μ. Since f on S cannot compute a linear function as $D > d + 1$, the circuit C must be contain at least one gate (even if its inputs are nontrivial linear functions themselves). Let A be a topmost (closest to the inputs) gate of C. If A computes a sum, then it can be replaced by an input (without increasing μ), so we can assume that A computes a product of some linear sums x and y, i.e., A computes $(x \oplus c_1)(y \oplus c_2) \oplus c_3$ where $c_1, c_2, c_3 \in \mathbb{F}_2$ are constants. In the following we assign either $x = c_1$ or $y = c_2$. This gives us an affine subspace of $\mathbb{F}_2^n$ of dimension at least $D - 1$ (if the dimension of the resulting subspace dropped to 0 this would mean that either x or y was constant on S contradicting the fact that the considered circuit was optimal). To proceed by induction we need to show that the substitution reduces μ by at least 4. For this, we consider two cases.

Case 1. Both x and y have outdegree 1.



We then assign $x = c_1$. This trivializes A to c_3, so all its successors are eliminated too. In total, we eliminate at least two gates (A and its successors) and at least two inputs (x and y). Hence μ is reduced by at least 4. (Note that A must have at least one successor as otherwise it would be the output gate, but this would mean that f was constant on an affine subspace of dimension at least d.)

Case 2. The outdegree of, say, x is at least 2.



Let B be another successor of x and let C be a successor of A. We assign $x = c_1$. This removes an input x and gates A, B, and C. If $B = C$ then C becomes a constant under the substitution (since both its inputs are constants) so its successors are also eliminated (note that C cannot be the output, because then our assumed disperser is trivialized by $x = c_1$, that is, it is constant on a subspace of dimension $D - 1 > d$). Thus, in this case we eliminate at least one input and at least three gates implying that μ is reduced by at least 4.

Plugging in an affine disperser for sublinear dimension in this argument gives a $3n - o(n)$ lower bound. It is also interesting to note that the inequality $G(C) + I(C) \geq 4(n - d - 1)$ is tight. To see this, note that the inner product function ($\mathrm{IP}(x_1, y_1, x_2, y_2, \ldots, x_{n/2}, y_{n/2}) = x_1y_1 \oplus x_2y_2 \oplus \cdots \oplus x_{n/2}y_{n/2}$) is an affine disperser for dimension $n/2 + 1$ (see, e.g., [12, Theorem A.1]) and has circuit size $n - 1$.

1.6 A $3.01n$ lower bound for affine dispersers

Our first main result is the following theorem.

Theorem 1.1. *The circuit size of an affine disperser for sublinear dimension is at least* $\left(3 + \frac{1}{86}\right) n - o(n)$.

The main ingredients of our proof are:

- the well-known gate elimination technique,
- affine substitutions that received limited attention previously,
- a restricted type of quadratic substitutions (read-once depth-2 quadratic sources) that has not been studied before,
- cyclic circuits, also a novel technique.

In the following, we describe these new ingredients and explain how we use them.

As one may have noticed from the gate elimination proof in the previous section, if one makes an appropriate substitution to an input of a non-linear binary gate (that is, not an XOR or equivalence gate), it makes this gate constant and therefore eliminates all the immediate descendents of this gate. This makes the induction step easier than in the case of a linear gate. On the other hand, linear gates, when stacked together, sometimes allow to reorganize the circuit. This idea has been used in [43, 57, 6, 52, 14]. Then affine restrictions can eliminate such gates while preserving the properties of an affine disperser.

Thus, it is natural to consider a circuit as composed of linear subcircuits connected by non-linear gates. If one considers a linear subcircuit and has to make an affine substitution, how to reorganize it efficiently? In our case analysis we do not just dive into an affine subspace: we make affine substitutions, 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 internal 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 [46, 17, 40, 45] (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 some 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 internal gates, we compute a linear combination of the number of internal gates and the number of bottlenecks. Such measures were previously considered by several authors. For example, Zwick [66] counted the number of (internal) gates minus the number of inputs of outdegree 1. The same measure was later used by Lachish and Raz [35] and by Iwama and Morizumi [25]. Kojevnikov and Kulikov [31] used a measure assigning different weights to linear and non-linear gates to show that Schnorr's $2n-O(1)$ lower bound [51] 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 [34, 30, 18].

Eventually, the wider class of substitutions one allows, the larger number of gates can be eliminated. Affine substitutions did not appear for us to be sufficient to make a progress, so we introduced a limited class of quadratic substitutions. Namely, we consider substitutions that form a *read-once depth-2 quadratic (rdq) source*, that is, a variable can be used in the right-hand side of a quadratic substitution only once, and only if it is not substituted by an affine or quadratic substitution itself. It extends the power of substitutions allowing us to handle our bottleneck cases, while leaves the possibility to consider the set of applied substitutions as a *source*, that is, a function turning the original inputs into new inputs. It turns out that an affine disperser for a sublinear dimension remains non-constant for read-once depth-2 quadratic sources of a sublinear dimension.

Of course, using such substitutions is restrictive (one must keep an eye on the limitations). However, by including the number of appropriate quadratic substitutions in our complexity measure, our amortized analysis accomplishes the desired lower bound.

1.7 A $3.11n$ lower bound for quadratic dispersers

The two considered functions, MOD_3^n and an affine disperser for dimension d, can be viewed as functions that are not constant on any sufficiently large set $S \subseteq \mathbb{F}_2^n$ that can be defined as the set of roots of k polynomials:

$$S = \{x \in \mathbb{F}_2^n \colon p_1(x) = p_2(x) = \cdots = p_k(x) = 0\}\,.$$

For MOD_3^n, $k \le n-2$ and each p_i is just a variable or its negation while for affine dispersers, $k \le n-d$ and p_i's are arbitrary linear polynomials. Note that the size of the set S, if it is nonempty, is at least $2^{n-\sum_k \deg p_k}$ by Warning's second theorem [61], and in our (linear) case the equality holds

$$|S| = 2^{n-k} \tag{1}$$

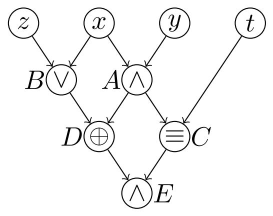
as long as p_i's are linearly independent.

A natural extension is to allow polynomials to have degree at most 2. The corresponding set S is called a quadratic variety. Formally, a function $f \in B_n$ is called an (n,k,s)-quadratic disperser if it is not constant on any variety of size at least s defined by at most k polynomials of degree at most 2. We prove the following result.

Theorem 1.2. *The circuit size of an $(n, 1.83n, 2^{o(n)})$-quadratic disperser is at least $3.11n$.*

Currently, explicit constructions of quadratic dispersers with such parameters are not known while showing their existence non-constructively is easy (see Lemma 2.2.1). Theorem 1.2 can be viewed as an additional motivation for their study.

We prove Theorem 1.2 by extending the gate elimination method. The proof goes by induction on the size of the current quadratic variety S. Note that for quadratic varieties the relation eq. (1) no longer holds:



$$\begin{aligned}
B &= (z \vee x)\\
A &= (x \wedge y)\\
D &= (B \oplus A)\\
C &= (A \equiv t)\\
E &= (D \wedge C)
\end{aligned}$$

Figure 1: An example of a circuit and the program it computes.

e.g., the set of roots of $n/2$ polynomials $x_1x_2 \oplus 1$, $x_3x_4 \oplus 1$, $\ldots$, $x_{n-1}x_n \oplus 1$ contains just one point. For this reason, we proceed as follows. We choose a polynomial p of degree 2 and consider two subvarieties of S: $S_0 = \{x \in S \colon p(x) = 0\}$ and $S_1 = \{x \in S \colon p(x) = 1\}$. We then estimate how much the size of the circuit shrinks for each of these varieties and how much the size of the variety shrinks. Roughly, we show that in at least one of these cases the circuit shrinks a lot while the size of the variety does not shrink a lot.

2 Definitions

2.1 Circuits

We consider binary circuits only and thus use the word "circuit" to denote a binary circuit. 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 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. For a circuit C, $G(C)$ is the number of internal gates and is also called the size of the circuit C. By $I(C)$ we denote the number of input gates. For a function $f \in B_{n,m}$, $C(f)$ is the minimum size of a circuit with n inputs and m outputs that computes f.

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. *In our diagrams, we denote all and-type gates by* $\wedge$*, and all xor-type gates by* $\oplus$*.*

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.

A simple example of a circuit is shown in Figure 1. For input gates, the corresponding variables are shown inside. For an internal gate, we show its operation inside and its label near the gate. As the figure shows, a circuit corresponds to a simple program for computing a Boolean function: each instruction of the program is a binary Boolean operation whose inputs are input variables or the results of the previous instructions.

In the remaining part of the paper, we show parts of a circuit. Such a part usually consists of a subset of gates and a subset of wires of a circuit. Thus, any gate shown in a picture may have an arbitrary number of additional outgoing wires. For some of the gates, we show their out-degree above the gate. Thus, a gate that has two out-going wires and a label 2 above it has no other out-going wires.

2.2 Affine and quadratic dispersers

Definition 2.2.1 (affine disperser)**.** An *affine disperser* for dimension $d(n)$ is a family of functions $f_n : \mathbb{F}_2^n \to \mathbb{F}_2$ such that for all sufficiently large n, f_n is non-constant on any affine subspace of dimension at least $d(n)$.

Definition 2.2.2 (quadratic variety)**.** A set $S \subseteq \mathbb{F}_2^n$ is called an (n,k)*-quadratic variety* if it can be defined as the set of common roots of $t \le k$ polynomials of degree at most 2:

$$S = \{x \in \mathbb{F}_2^n : p_1(x) = \cdots = p_t(x) = 0\}$$

where p_i is a polynomial of degree at most 2, for each $1 \le i \le t$.

Definition 2.2.3 (quadratic disperser)**.** A Boolean function $f \in B_n$ is called an (n,k,s)*-quadratic disperser* if f is non-constant on any (n,k)-quadratic variety $S \subseteq \mathbb{F}_2^n$ of size at least s. Here k and s can be functions of n.

The following lemma shows that almost all functions from B_n are $(n, 2^{o(n)}, 2^{o(n)})$-quadratic dispersers.

Lemma 2.2.1. *Let $\omega(kn^2) \le s \le 2^{o(n)}$. Let $D_n \subseteq B_n$ be the set of (n,k,s)-quadratic dispersers. Then $\frac{|D_n|}{|B_n|} \to 1$ when $n \to \infty$.*

Proof. There are $q = \frac{n(n+1)}{2} + 1 = \Theta(n^2)$ multilinear monomials of degree at most 2 in $\mathbb{F}_2[x_1, \ldots, x_n]$. Therefore, there are 2^q polynomials of degree at most 2, and at most 2^{qk} (n,k)-quadratic varieties. Each function that is not an (n,k,s)-quadratic disperser can be specified by

1. an (n,k)-quadratic variety, where it takes a constant value,
2. this value (0 or 1),
3. values at the remaining at most $2^n - s$ points.

Thus, the number of functions that are not (n,k,s)-quadratic dispersers is bounded from above by $2^{qk} \cdot 2 \cdot 2^{2^n - s} = 2^{2^n} 2^{qk+1-s} = 2^{2^n} 2^{-\Theta(s)} = o(|B_n|)$. □

2.3 Generalizations of circuits

Cyclic circuits In this paper we apply a sequence of transformations on circuits. To accomodate this we use the following generalization of circuits (note that we do *not* use xor-inputs circuits that have been introduced solely for demonstration purposes in Subsection 1.5). These generalized circuits may contain specific types of cycles where all 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 as 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; for the case of a cyclic xor-circuit all these systems are linear and 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). An example of a fair cyclic xor-circuit is shown in Figure 2. 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).

Relationship between cyclic and acyclic xor-circuits It is not difficult to show that for multiple outputs, fair cyclic xor-circuits form a stronger model than acyclic xor-circuits. For example, the 9 functions computed simultaneously by the cyclic xor-circuit shown in Figure 2 cannot be computed by an acyclic xor-circuit with 9 gates. To see this, assume for the sake of contradiction, that an acyclic xor-circuit with 9 gates computes the same functions. Since the circuit has 9 gates all gates must compute outputs. Consider a topologically minimal gate G. Such a gate exists since the circuit is acyclic. Since G is topologically minimal it computes the sum of two input gates, therefore it cannot compute any output.

On the other hand, in the case of single-output circuits, cycles do not make the model stronger: a minimal xor-circuit of k variables computing a single output has exactly $k-1$ internal gates and is acyclic.

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. All input nodes belong to X, and 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.

3 Lower bound $3.01n$ for affine dispersers

This section is devoted to the proof of a $(3+\frac{1}{86})n - o(n)$ lower bound on the circuit size of affine dispersers. ***In this section, we abuse the notation by using the word "circuit" to mean a fair semicircuit.***

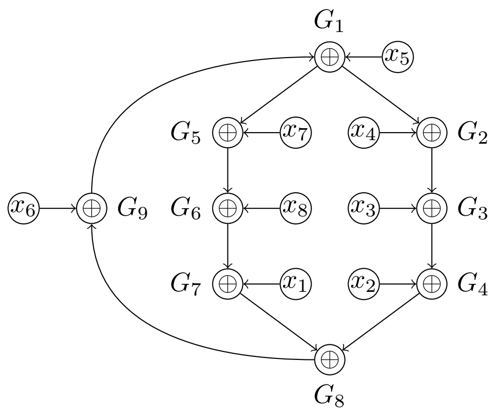
The proof goes by induction. We start with an affine disperser and a circuit computing it on $\{0,1\}^n$. Then we gradually shrink the space where it is computed by adding equations ("substitutions") for variables. This allows us to simplify the circuit by reducing the number of internal gates (and other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

In Subsection 3.1, we formally define a *read-once depth-two quadratic source* (a source of special type arising from constant, affine, and quadratic equalities) that we later use instead of an affine source. We prove that a disperser for affine sources is also a disperser for our generalized sources, and thus known constructions of affine dispersers can be used for our needs.

In Subsection 3.2, we define the circuit complexity measure that allows us to perform "amortized" analysis. We need to define the notion of a so-called troubled gate here. Informally speaking, this is a gate participating in a specific type of combinatorial fragment of a circuit that is a "bottleneck" for the proof: namely, it 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.

The main result formulated in this section is the following: 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 internal gates, the number of troubled gates, the number of quadratic equalities, and the number of inputs. The optimal values of coefficients in this linear combination come from solving a simple linear program.

Eventually, Subsection 3.3 employs all the developed techniques in order to prove the main lower bound of this section modulo technicalities: substitution and transformation rules, and the case distinction argument.



$$
\begin{aligned}
G_1 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_2 &= x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_3 &= x_1 \oplus x_2 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_4 &= x_1 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_5 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_8 \\
G_6 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \\
G_7 &= x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \\
G_8 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_8 \\
G_9 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_7 \oplus x_8
\end{aligned}
$$

$$
\begin{aligned}
G_1 &= G_9 \oplus x_5 \\
G_2 &= G_1 \oplus x_4 \\
G_3 &= G_2 \oplus x_3 \\
G_4 &= G_3 \oplus x_2 \\
G_5 &= G_1 \oplus x_7 \\
G_6 &= G_5 \oplus x_8 \\
G_7 &= G_6 \oplus x_1 \\
G_8 &= G_4 \oplus G_7 \\
G_9 &= G_8 \oplus x_6
\end{aligned}
\qquad
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}
\times
\begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \\ G_6 \\ G_7 \\ G_8 \\ G_9 \end{bmatrix}
=
\begin{bmatrix} x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_7 \\ x_8 \\ x_1 \\ 0 \\ x_6 \end{bmatrix}
$$

Figure 2: An example of a cyclic xor-circuit that is more efficient than an ordinary circuit. In this case all the gates are labeled with $\oplus$. The affine functions computed by the gates are shown below the circuit. The bottom row shows the program computed by the circuit as well as the corresponding linear system.

The details of transformation rules used to normalize and simplify the circuit are given in Subsection 3.4.

The simplest substitutions are replacing a variable by a constant or by an already computed function of other variables. After defining them formally, we show how to make substitutions in fair semicircuits, and how to normalize them afterwards. We introduce five normalizing transformations 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 proceed to a more complicated case: 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.

In order to balance our complexity measure, we need to analyze carefully how many new troubled gates can be introduced by applying a normalizing transformation. 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 a rather large dimension where the circuit outputs a constant). This implies that the bottleneck case cannot appear too often during the gate elimination process.

Eventually, the complete case distinction argument for eliminating a part of the circuit of the required measure is given in Subsection 3.5.

3.1 Read-once depth-2 quadratic sources

We generalize affine sources as follows.

Definition 3.1.1. 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 $i, k \in F$ $(i \neq k)$ and c_i, c_k, c_j are constants; every variable from F appears in the right-hand of at most one quadratic equation;

- 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 a 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*.

An example of such a system is shown in Figure 3. In particular, the figure explains the name: the right-hand side of each equality is computed by a *depth-two* circuit (with binary $\wedge$-gates and unbounded fan-in $\oplus$-gates) and computes a Boolean function of *degree at most two*; also, each variable is *read* by an $\wedge$-gate at most *once*.

In the proof of the lower bound, we gradually build a straight-line program as follows. By analyzing the structure of the current circuit, we find a free variable x_j and a function p (of degree at most 2) so that:

1. Many gates become redundant in the circuit under the substitution $x \leftarrow p$. In particular, informally x becomes disconnected from the circuit. (The formal details of transformations of circuits after the substitutions are delayed to Subsection 3.4.)

2. Adding the equality $x = p$ to the current rdq-source keeps it an rdq-source. In particular, x stops to be free.

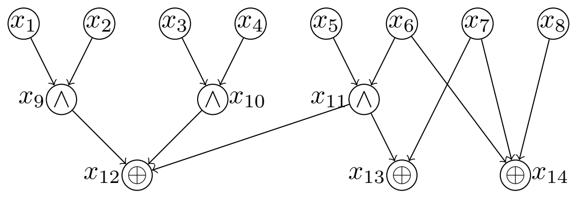


Figure 3: An example of an rdq-source. Note that a variable can be read just once by an and-type gate while it can be read many times by xor-type gates.

Thus, a substitution is written as $x \leftarrow p$ and is used to shrink a circuit, where an equality is written as $x = p$ and is used to shrink the source. Below, we state formally possible ways to further restrict an rdq-source.

Lemma 3.1.1. *Let $R \subseteq \mathbb{F}_2^n$ be an rdq-source of dimension d specified by sets F, L, Q and a system of equalities. Let $x_j \in F$ be a free variable of R. Then, adding any of the following equalities to R results in an rdq source of dimension $d-1$.*

1. *$x_j = c$, where $c \in \{0,1\}$;*
2. *$x_j = \bigoplus_{i \in I} x_i \oplus c$, where $I \subseteq F \cup Q$ and $c \in \{0,1\}$, provided x_j is unprotected;*
3. *$x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j$, where $i, j, k \in F$ and $c_i, c_k, c_j \in \{0,1\}$, provided that x_j is unprotected;*
4. *$x_j = x_i \oplus c$, where $c \in \{0,1\}$ and both x_i and x_j are protected free variables appearing in the same quadratic equality in R.*

Proof. After adding a new equality, the variable x_j is not free anymore. For each of the four possibilities, we show how to rewrite the existing equalities of R so that the properties of an rdq-source still hold.

1. We move j from F to L and replace every occurrence of x_j by c in all (affine and quadratic) equalities of R. If x_j appears in the right-hand side of a quadratic equality, $x_j = c$ makes it affine. We then move the corresponding variable from Q to L.
2. We move j from F to L. Since x_j is unprotected, it does not appear in the right-hand side of quadratic equalities. In (the right-hand side of) all affine equalities, we replace x_j by $\bigoplus_{i \in I} x_i \oplus c$. This leaves them affine.
3. We move j from F to Q. Since x_j is unprotected, it does not appear in quadratic equalities. We do nothing with affine equalities as they are allowed to contain variables from Q.
4. We move j from F to L. In every affine equality, we replace x_j by $x_i \oplus c$. The only quadratic equality involving x_j becomes affine. The variable x_i becomes unprotected.

The dimension of the rdq-source drops by one, as we remove exactly one variable from F. □

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.

Definition 3.1.2. 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. The function 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.

Definition 3.1.3. An *rdq-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, for every rdq-source R of dimension at least $d(n)$, $f_n|_R$ is non-constant.

Now we show that affine dispersers are also rdq-dispersers for related parameters, allowing us to prove a lower bound on the complexity of affine dispersers by using the fact that they work as rdq-dispersers as well.

Lemma 3.1.2. *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.*

Proof. We prove this lemma by showing that every rdq-sourse $R \subseteq \mathbb{F}_2^n$ of dimension d contains an affine subspace of dimension at least $d/2$.

For each quadratic equality $x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j$ of R, further restrict R by adding an equality $x_i = 0$. This replaces a quadratic equality by two affine equalities $x_i = 0$ and $x_j = c_i(x_k \oplus c_k) \oplus c_j$; the number of free variables decreases by one. Also, since the free variables do not occur in the left-hand side, the newly introduced affine substitution is consistent with the previous affine substitutions.

Since the variables occurring on the right-hand side of quadratic equalities are disjoint, we have initially that $2|Q| \leq |F| = d$, so the number of newly introduced affine equalities is at most $d/2$. $\square$

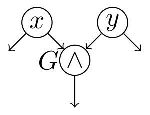
Note that it is important in the proof that protected variables do not appear in the left-hand side. The proposition above is obviously false for quadratic *varieties*: no Boolean function can be non-constant on all sets of common roots of $n - o(n)$ quadratic polynomials. For example, the system of $n/2$ quadratic equalities $x_1x_2 = x_3x_4 = \ldots = x_{n-1}x_n = 1$ defines a single point, so any function is constant on this set.

3.2 Circuit complexity measure

Before we introduce our complexity measure, we define an important notion: *troubled gate*. Such gates correspond to "bottlenecks" in our case analysis in Subsection 3.5, thus the number of them is important for the measure.

Definition 3.2.1 (troubled gates)**.** We say that an internal gate G is *troubled* if it satisfies the following three criteria:

- G is an and-type gate of outdegree 1,
- the gates feeding G are input gates,
- both input gates feeding G have outdegree 2.



Recall that we denote all and-type gates by $\wedge$, and all xor-type gates by $\oplus$.

For a circuit C and an rdq-source $R \subseteq \mathbb{F}_2^n$, 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 equalities 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 equality in R). The constants $\alpha_Q, \alpha_T, \alpha_I > 0$ will be chosen later.

We will show later (Proposition 3.4.3) that when a gate is removed from a circuit by applying a normalizing transformation 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 3.2.1. *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 $\frac{n}{2} + \frac{5d}{2}$.*

Proof. Let V be the set of the inputs, $|V| = n$. Assume to the contrary that $t \geq \frac{n}{2} + \frac{5d}{2}$. Let v_i be the number of variables feeding exactly i troubled gates. Since a variable feeding a troubled gate must have outdegree 2, $v_i = 0$ for $i > 2$. By double counting the number of wires from inputs to troubled gates, $2t = v_1 + 2v_2$. Since $v_1 + v_2 \leq n$,

$$n + 5d \leq 2t = v_1 + 2v_2 \leq n + v_2 \tag{2}$$

To derive a contradiction, we construct an affine subspace S of $\mathbb{F}_2^n$ of dimension at least d such that C is constant on S. To do this, start with empty sets L and U. The set L will be populated by affine equations defining the subspace S, the set U will contain variables that will remain unconstrained by L. Denote by T the set of inputs that feed two troubled gates. By eq. (2), $|T| = v_2 \geq 5d$. Repeat the following step while T is not empty.

Take any input $x \in T$, it feeds two troubled gates G_1 and G_2. Denote other variables feeding G_1 and G_2 by y_1 and y_2. In what follows, we call two inputs feeding the same troubled gate *neighbors*. Consider two cases.

1. $y_1 \neq y_2$. The gates G_1 and G_2 compute operations $(x \oplus a_1)(y_1 \oplus b_1) \oplus c_1$ and $(x \oplus a_2)(y_2 \oplus b_2) \oplus c_2$. Note that if $y_1 = b_1$ and $y_2 = b_2$, then the function computed by the circuit does not depend on x: both gates that are fed by x are constant under $y_1 = b_1$ and $y_2 = b_2$. Each of y_1 and y_2 has at most one other neighbor in C. We remove x, y_1, y_2 and all neighbors of y_1, y_2 (at most five variables in total) from T, we add x to U, and we add equalities $y = b_1$ and $y = b_2$ to L.
2. $y_1 = y_2$. Denote them by y. The gates G_1 and G_2 compute operations
$$(x \oplus a_1)(y \oplus b_1) \oplus c_1 \text{ and } (x \oplus a_2)(y \oplus b_2) \oplus c_2\,.$$
If $a_1 = a_2$, then C does not depend on x under $y = a_1$. We update T, U, L as follows: $U = U \cup \{x\}$, $T = T \setminus \{x, y\}$, $L = L \cup \{y = a_1\}$. The case $b_1 = b_2$ is treated similarly. Assume now that $a_1 = a_2 \oplus 1$ and $b_2 = b_1 \oplus 1$. Hence, G_1 and G_2 compute operations
$$(x \oplus a_1)(y \oplus b_1) \oplus c_1 \text{ and } (x \oplus a_1 \oplus 1)(y \oplus b_1 \oplus 1) \oplus c_2\,.$$
Consider an equality $y = x \oplus a_1 \oplus b_1 \oplus 1$. For every assignment satisfying this equality, C does not depend on x as both G_1 and G_2 are constant. We update T, U, L as follows: $U = U \cup \{x\}$, $T = T \setminus \{x, y\}$, $L = L \cup \{y = x \oplus a_1 \oplus b_1 \oplus 1\}$.

At every step, we remove no more than five variables from T and add one variable to U. Since $|T| \geq 5d$ initially, U contains at least d variables in the end of the process. The system of equalities L is consistent. Indeed, at every step, we add an equality to L that has in the left-hand side a variable whose neighbor is from T. Right after this, we remove both variables from T, so that no future iteration will try to assign anything to them.

Thus, the circuit is constant on the affine subspace defined by the following equalities:

- all equalities from L;
- equalities $z = 0$ for all variables except for those from U and those from the left-hand side of equalities from L.

Since the equalities from L make the circuit independent of all variables from U, the resulting affine subspace has dimension at least d. $\square$

We are now ready to state a lower bound on the circuit complexity measure of rdq-dispersers.

Theorem 3.1. *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 (thus, C computes f on an rdq-source $R = \mathbb{F}_2^n$ defined by an empty straight-line program). Let $0 \le \alpha_Q, \alpha_I$, $0 \le \alpha_T \le 1/4$ be constants. Then $\mu(C, \mathbb{F}_2^n) \ge \delta \cdot (n-d-2)$ where the constant δ is defined as follows:*

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

and

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

We defer the proof of this theorem to Subsection 3.5. This theorem, together with Lemma 3.1.2, implies a lower bound on the circuit complexity of affine dispersers.

Corollary 3.2.1. *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*

$$\left(\delta - \frac{\alpha_T}{2} - \alpha_I\right) n - o(n).$$

Proof. Note that $q = 0$, $i \le n$, $t < \frac{n}{2} + \frac{5d}{2}$ (see Lemma 3.2.1). Thus, the circuit size is

$$\begin{aligned} g &= \mu - \alpha_Q \cdot q - \alpha_T \cdot t - \alpha_I \cdot i \\ &> \delta(n-2d-2) - \alpha_T \cdot \left(\frac{n}{2} + \frac{5d}{2}\right) - \alpha_I \cdot n \\ &= \left(\delta - \frac{\alpha_T}{2} - \alpha_I\right) n - \left(2\delta + \frac{5\alpha_T}{2}\right) d - 2\delta \\ &= \left(\delta - \frac{\alpha_T}{2} - \alpha_I\right) n - o(n). \end{aligned}$$

□

The maximal value of $\delta - \frac{\alpha_T}{2} - \alpha_I$ is given by the following linear program: maximize $\delta - \frac{\alpha_T}{2} - \alpha_I$ subject to

$$\begin{aligned} \beta + 4\alpha_T &= 1 \\ \alpha_T, \alpha_Q, \alpha_I, \beta &\ge 0 \\ \delta &\le \alpha_I + \min\left\{\frac{\alpha_I}{2}, 4\beta, 3+\alpha_T, 2\beta+\alpha_Q, 5\beta-\alpha_Q, 2.5\beta+\frac{\alpha_Q}{2}\right\}. \end{aligned}$$

The optimal values for this linear program are

$$\begin{aligned} \alpha_T &= \frac{1}{43}, \\ \alpha_Q &= 1 + 22\alpha_T = \frac{65}{43}, \\ \alpha_I &= 6 + 2\alpha_T = 6 + \frac{2}{43}, \\ \beta &= 1 - 4\alpha_T = \frac{39}{43}, \\ \delta &= 9 + 3\alpha_T = 9 + \frac{3}{43}. \end{aligned}$$

This gives the main result of this section.

Theorem 1.1. *The circuit size of an affine disperser for sublinear dimension is at least $\left(3 + \frac{1}{86}\right) n - o(n)$.*

3.3 Gate elimination

In order to prove Theorem 3.1 we first show that it is always possible to make a substitution and decrease the measure by δ.

Theorem 3.2. *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 a 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*

$$\mu(C', R') \leq \mu(C, R) - \delta(s - s')\,.$$

Before we proceed to the proof, we show how to derive the main theorem from this claim:

Proof of Theorem 3.1. We prove that, for any rdq-source $R \subseteq \mathbb{F}_2^n$ of dimension s and any C computing $f|_R$, it holds that $\mu(C,R) \geq \delta(s-d-2)$. We do this by induction on s. Note that the statement is vacuously true for $s \leq d+2$, since μ is non-negative. Now suppose the statement is true for all rdq-sources of dimension strictly less than s for some $s > d+2$, and let R be an rdq-source of dimension s. Let C be a fair semicircuit computing $f|_R$. Let R' be the rdq-source of dimension s' whose existence is guaranteed by Theorem 3.2, and let C' be a fair semicircuit computing $f|_{R'}$. We have that

$$\mu(C,R) \geq \mu(C', R') + \delta(s - s') \geq \delta(s - d - 2),$$

where the second inequality holds by the induction hypothesis. $\square$

We have just proved our main theorem, and we proceed to proving the statements whose proofs we deferred.

3.4 Cyclic circuit transformations

3.4.1 Basic substitutions

In this section we consider several types of substitutions. Substitutions transform circuits into simpler (yet not equivalent) ones.

Substituting an input with a constant is straightforward.

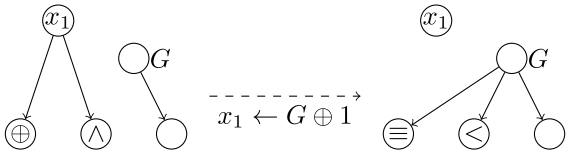
Substitution by a constant. Let C be a circuit with input gates $x_1, \ldots, x_n$, and let $c \in \{0,1\}$ be a constant. To make a substitution $x_1 \leftarrow c$, for every gate G fed by x_1, replace the operation $g(x_1, t)$ computed by G with the operation $g'(x_1, t) = g(c, t)$ and remove a wire from x_1 to G.

Proposition 3.4.1. *Substitution of a variable x_1 by a constant c transforms a circuit C into another circuit C' (in particular, it is still a fair semicircuit) that has the same number of internal 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 $h(c, x_2, \ldots, x_n)$.*

The substitution where an input x is replaced by an output of a different gate G is more complicated. In this case, in each gate fed by x, we replace the wires coming from x by the wires coming from G.

Substitution by a function. Let C be a circuit with input gates $x_1, \ldots, x_n$, let $g(x_2, \ldots, x_n)$ be a function computed by a gate G unreachable from x_1 by a directed path in C, and let $c \in \{0,1\}$. To make a substitution $x_1 \leftarrow G \oplus c$, for every gate H fed by x_1, replace a wire from x_1 to H by a wire from G to H; if $c = 1$, negate the corresponding operand in the operation computed by H.

The picture below gives an example. We replace x_1 by the negation of (the function computed at) G. The operation $x \oplus y$ computed at one of the successors of x_1 is replaced by $x \equiv y = x \oplus y \oplus 1$. The operation $x \wedge y$ computed at the other successor is replaced by $x < y = \overline{x} \wedge y$.



Proposition 3.4.2. *Substitution by a function $g(x_2, \ldots, x_n)$ for a variable x_1 transforms a circuit C into a circuit C' that is still 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 $h(g(x_2, \ldots, x_n), x_2, \ldots, x_n)$.*

This substitution may introduce a new troubled gate only in the following case $out(x_1) = 1$, x_1 feeds an and-gate, and G is an input 1-gate.

Proof. Note that we require that G is not reachable from x_1 (thus we do not introduce new cycles), and also that g does not depend on x_1. Functions computed in the gates are the solution of the system corresponding to the circuit (see Subsection 2.3). The transformation simply replaces every equation of the form $H = F \odot x_1$ with the equation $H = F \odot G$ (and every equation of the form $H' = x_1 \odot x_1$ with the equation $H' = G \odot G$).

In order to prove that C' is a fair semicircuit, we show that for each assignment to the inputs, there is a unique assignment to the gates of C' that is consistent with the inputs. Consider specific values for $x_2, \ldots, x_n$ and the value $g(x_2, \ldots, x_n)$ for x_1. Assume that the solution for the old system does not satisfy the new equation. Then it violates the corresponding equation in the old system, a contradiction. Vice versa, consider two different solutions for the new system. Both of them (augmented with $x_1 = g(x_2, \ldots, x_n)$) must satisfy the old system, but the old system has a unique solution.

The only gates that could potentially become troubled are the successors of x_1. Let A be such a gate. After the substitution, it is fed by G. Hence, A may become troubled only if $G = x_i$ is an input gate and $out(G) = 2$ after the substitution. Since after the substitution, the outdegree of G is increased by $out(x_1)$, we conclude that before the substitution it holds that $out(x_1) = out(G) = 1$. □

In what follows, however, we will also use substitutions that do not satisfy the hypothesis (G is unreachable from x_1) of this proposition: substitutions that create cycles. We defer this construction to Subsection 3.4.3.

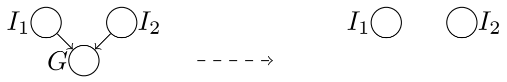
3.4.2 Normalization

In order to work with a circuit, we are going to assume that it is "normalized", that is, it does not contain obvious inefficiencies (such as trivial gates, etc.), in particular, those created by substitutions. We describe certain normalizing transformations 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, troubled gates, see Definition 3.2.1).

In order to count the number of troubled gates, we limit normalizing transformations of the circuit to a few very specific cases described below and make sure that we never get more than four new troubled gates per eliminated gate.

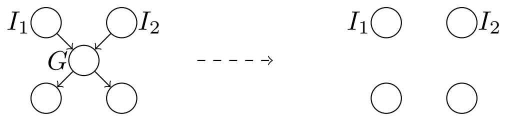
We say that a circuit is *normalized* if none of the following transformations is applicable to it. Each transformation eliminates a gate G whose inputs are gates I_1 and I_2. (Note that I_1 and I_2 can be inputs or internal nodes, and, in the case where a cyclic part has been modified, they can coincide with G itself.)

Transformation 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.

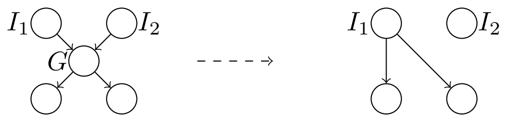
Transformation 2: If G is trivial, i.e., it computes a constant function c of the circuit inputs (not necessarily a constant operation on the two inputs of G), remove G and "embed" this constant to the next gates. That is, for every gate H fed by G, replace the operation $h(g,t)$ computed in this gate (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 Transformation 3 or 2 to its successors.)



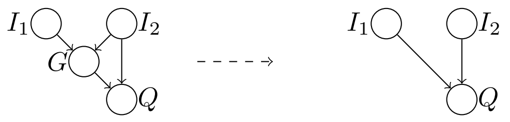
Transformation 3: If G is passing, i.e., it computes an operation depending only on one of its inputs, remove G by reattaching its outgoing wires to that input. This may also require changing the operations computed at its successors (the corresponding input may be negated; note that an and-type gate (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 Transformation 1). In this special case, we remove G and mark the corresponding input of G (or its negation) as the output gate.



Transformation 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*.



Transformation 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 Transformation 3 or 2.

Proposition 3.4.3. *Each of the Transformations 1–5 removes one internal gate, and 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 transformations change the functions of n input variables computed in the gates that are not removed. A fair semicircuit remains a fair semicircuit.*

Proof. Fairness. The circuit remains fair since no transformation changes the set of solutions of the system.

New troubled gates. For all the transformations, the only gates that may become troubled are I_1, I_2 (if they are and-type gates), and the gates they feed after the transformation (if I_1 or I_2 is a variable). Note that other gates cannot generate troubled gates during a particular transformation (they can do it in the next transformation, but this Proposition considers a *single* transformation). In Transformation 2 they may become constant (thus not and-type) and could change the degree only of a disappearing gate. In Transformations 3 and 4 they are fed by I_1, I_2, so if they become troubled themselves, this is already counted for I_1, I_2; these transformations do not change degrees of any other gate. Transformation 5 just replaces an operation with a unary one, so all troubled gates appear in the subsequent Transformation 2 or 3.

It is clear from the figures that during a single transformation each of I_1, I_2 may create at most two new troubled gates: troubled gates appear if

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

- *an and-type gate changes its outdegree from 2 to 1,*
- *an input of the circuit changes its outdegree to 2,*
- *an and-type gate becomes feeded by an input of the circuit.*

Hence each transformation, when applied, introduces at most four new troubled gates.

(A typical case where exactly four troubled gates are introduced is where I_1, I_2 are circuit inputs, we change their outdegrees to two, and their descendants are distinct and-type gates of indegree one and outdegree one.) $\square$

3.4.3 Affine substitutions

In this section, we introduce more complicated substitutions that do create cycles. This will be needed in order to perform affine substitutions.

Our particular goal is to apply a substitution of the form $x_1 \leftarrow \bigoplus_{i\in I} x_i \oplus c \oplus b$ without computing its right-hand side explicitly. For this, we replace 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) by a trivial gate computing a constant $b \in \{0,1\}$, while x_1 is replaced by an internal gate computing the function $\bigoplus_{i\in I} x_i \oplus c \oplus b$, when the subcircuit for computing this function is already present in the circuit. The details of this transformation are given in Lemma 3.4.2.

In order to prove a formal statement about the result of this substitution, we need to generalize the terminology to xor-circuits and prove a structural lemma (which would be trivial for acyclic circuits).

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 immediate predecessors (arguments) 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. In the following lemma we show that it is always possible to find such a path in a fair cyclic circuit too. However, the functions computed in gates of a cyclic circuit result as a solution of a system of equations and are not just consequent compositions like it is in the case of ordinary circuits, thus it may be possible that some nodes on this path do not depend on x. (For example, in Figure 2, gate G_4 is fed by x_2 but does not depend on it.)

Lemma 3.4.1. *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.*

Proof. Let $\mathcal{R}$ be the set of internal gates that are reachable from x, and $\mathcal{U}$ be the set of internal gates that are not reachable from x. Let us enumerate the gates in such a way that gates from $\mathcal{U}$ have smaller indices than gates from $\mathcal{R}$. Then the circuit C corresponds to the system

$$\begin{bmatrix} U & 0 \\ R_1 & R_2 \end{bmatrix} \times \mathcal{G} = \begin{bmatrix} L_U \\ L_R \end{bmatrix},$$

where $\mathcal{G} = (g_1, \ldots, g_{|C|})^T$ is the vector of unknowns (the gates' values), U is the principal submatrix corresponding to $\mathcal{U}$ (a square submatrix whose rows and columns correspond to the gates from $\mathcal{U}$). Note that the upper right part of the matrix is 0, because there are no wires going from $\mathcal{R}$ to $\mathcal{U}$.

Since the value of G depends on the input x, there is an assignment α to all other inputs such that G's value is different in the two cases where (L_U, L_R) corresponds to $(\alpha, x = 0)$ and to $(\alpha, x = 1)$. However, L_U does not change when the value of the single input x changes (as x does not appear in this vector of constants; recall that it also includes the inputs).

Note that the submatrix U must be non-singular, because otherwise the whole matrix would be singular, which would contradict the fairness of C, so $U \times \mathcal{G}' = L_U$ determines the values of $\mathcal{G}'$ uniquely. Therefore, the values of unreachable gates are uniquely determined by α and do not depend on x. $\square$

We now come to rewiring. Let G be a gate computing $x_1 \oplus \bigoplus_{i\in I} x_i \oplus c$ and not reachable by a directed path from any and-type gate, and let $b \in \{0,1\}$ be a constant. In order to perform a substitution $x_1 =$

$\bigoplus_{i\in I} x_i \oplus c \oplus b$, we define the affine substitution transformation of a circuit (see Figure 4). Consider a path from x_1 to G that is guaranteed to exist by Lemma 3.4.1. Denote the internal gates on this path by $G_1,\dots,G_k = G$. Denote by $T_1,\dots,T_k$ the other inputs of these gates. Note that we assume that $G_1,\dots,G_k$ are pairwise different gates while some of the gates $T_1,\dots,T_k$ may coincide with each other and with some of $G_1,\dots,G_k$ (it might even be the case that $T_i = G_i$).

The affine substitution transformation adds a new internal gate Z. to the circuit. For every $i > 1$, the wire from T_i to G_i is replaced by a wire from T_i to G_{i-1}, and the wire from T_1 to G_1 is replaced by a wire from T_1 to Z. For every $i \ge 1$, the wire from G_i to G_{i+1} is replaced by a wire from G_{i+1} to G_i, and the wire from x_1 to G_1 is replaced by a wire from G_1 to Z.

An example of the affine substitution transformation is shown in Figure 4 (where $k = 4$ for the sake of example). The gates $A_0,\dots,A_k$ are shown on the picture just for convenience: any of $x_1, Z, G_1,\dots,G_k$ may feed any number of gates, not just one A_i. In the following lemma we summarize important properties of this transformation.

Lemma 3.4.2. *Let C be a fair semicircuit with input gates $x_1,\dots,x_n$ and internal gates $G_1,\dots,G_m$. 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 $I \subseteq \{2,\dots,n\}$. Let $b \in \{0,1\}$ be a constant. Then the affine substitution transformation applied to C results in a new circuit C' with the following properties:*

1. *C' has the same gates as C, plus a new internal gate Z; 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 $f(x_1,\dots,x_n)$ is the function computed by an internal gate in C and $f'(x_2,\dots,x_n)$ is the function computed by its counterpart in C', then $f(\bigoplus_{i\in I} x_i \oplus c \oplus b, x_2,\dots,x_n) = f'(x_2,\dots,x_n)$. The gate Z computes the function $\bigoplus_{i\in I} x_i \oplus c \oplus b$ (which on the affine subspace equals x_1).*

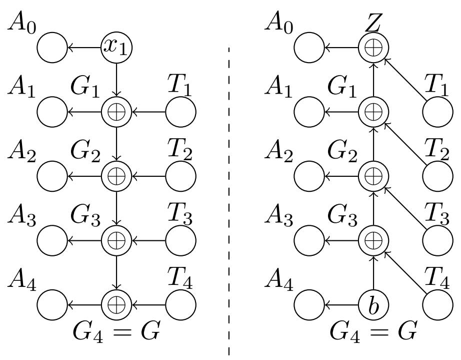
Proof. The first four items of the lemma follow from the definition of the transformation.

To show the fairness of C' (the fifth item), assume the contrary, that is, the sum of a subset S of rows of the new matrix is zero. There are two differences between the programs corresponding to linear parts of C and C': instead of $G_1 \oplus T_1 = x_1$, C' contains $G_1 \oplus T_1 \oplus Z = 0$; additionally, C' contains $G_k = b$. If the set of rows S contains the row corresponding to $G_1 \oplus T_1 \oplus Z = 0$, then the same rows of C (without the Z-column) also sum up to zero. Hence, S must contain the row for $G_k = b$. However, this would mean that if we sum up the corresponding equations for C, we get $G_k = \mathrm{const} \oplus \bigoplus_{j\in J} x_j$ where $J \not\ni 1$ (note that x_1 was replaced by Z in the new system, and cancelled out by our assumption). This contradicts the assumption of the lemma that G_k computes the function $x_1 \oplus \bigoplus_{i\in I} x_i \oplus c$. Therefore, the matrix for C' has full rank.

The programs shown next to the circuits explain that for $x_1 = \bigoplus_{i\in I} x_i \oplus c \oplus b$, the gates $G_1,\dots,G_k$ compute the same values in C' and C. Indeed, $G = G_k$ in C computes $x_1 \oplus \bigoplus_{i\in I} x_i \oplus c$ (by the statement of the lemma). From the structure of C, we see that this is $x_1 \oplus \bigoplus_{i=1}^k T_i$, i.e., $\bigoplus_{i=1}^k T_i = \bigoplus_{i\in I} x_i \oplus c$), which is b under $x_1 = \bigoplus_{i\in I} x_i \oplus c \oplus b$ (and it is set to b in C'). Then, looking at the systems shown next to the circuits and summing up the lines $G_i = G_{i-1} \oplus T_i$ (resp., $G_{i-1} = G_i \oplus T_i$) and using $G_k = b$ we see that for $i \ge 2$, $G_i = \bigoplus_{j=i+1}^k T_j$ in both cases. Then summing up the lines we also see that G_1 computes $G_k \oplus \bigoplus_{i=2}^k T_i$ in C and G_1 computes $b \oplus \bigoplus_{i=2}^k T_i$ in C'. Since $Z = G_1 \oplus T_1$ in C', the value of Z is $G_1 \oplus T_1 = b \oplus \bigoplus_{i=1}^k T_i$ is also clearly correct. ☐

Corollary 3.4.1. *The affine substitution transformation does not introduce new troubled gates.*

$$
\begin{aligned}
G_1 &= x_1 \oplus T_1\\
G_2 &= G_1 \oplus T_2\\
G_3 &= G_2 \oplus T_3\\
G_4 &= G_3 \oplus T_4
\end{aligned}
$$



$$
\begin{aligned}
Z &= G_1 \oplus T_1\\
G_1 &= G_2 \oplus T_2\\
G_2 &= G_3 \oplus T_3\\
G_3 &= G_4 \oplus T_4\\
G_4 &= b
\end{aligned}
$$

$$
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1
\end{bmatrix}
\times
\begin{bmatrix} G_1\\ T_1\\ G_2\\ T_2\\ G_3\\ T_3\\ G_4\\ T_4 \end{bmatrix}
=
\begin{bmatrix} x_1\\ 0\\ 0\\ 0 \end{bmatrix}
$$

$$
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}
\times
\begin{bmatrix} G_1\\ T_1\\ G_2\\ T_2\\ G_3\\ T_3\\ G_4\\ T_4\\ Z \end{bmatrix}
=
\begin{bmatrix} 0\\ 0\\ 0\\ 0\\ b \end{bmatrix}
$$

Figure 4: Transformation from Lemma 3.4.2. We use $\oplus$ as a generic label for xor-type gates. That is, in the picture, gates labelled $\oplus$ may compute the function $\equiv$.

Proof. In C', the gates $Z, G_1, \ldots, G_{k-1}$ are xor-type; the operation computed by G_k is constant; the gate A_0 is fed by a xor-type gate; for the gates $T_1, \ldots, T_k$ we don't change their type nor their inputs. All other gates are the same as in C. $\square$

After we apply the transformation, we apply Transformation 2 to G. Since the only troubled gates introduced by this transformation are the inputs of the removed gate, no troubled gates are introduced (and one gate, G itself, is eliminated, thus the combination of Lemma 3.4.2 and Transformation 2 does not increase the number of internal gates). Note also that even if G computes $x_1 \oplus c$ (i.e., $I = \emptyset$), it is still an internal gate, so that the gates it feeds cannot be troubled.

3.5 Gate elimination: Full proof

Let C be a circuit. By normalize(C) we mean a circuit resulting from the following procedure: while at least one of the Transformations 1–5 is applicable to C, apply the first such transformation. We say that C is *normalized*, if none of of the transformation rules is applicable to C (in other words, $C = \text{normalize}(C)$).

Lemma 3.5.1. *Let C be a circuit computing $f|_R$ for a function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ and an rdq-source $R \subseteq \mathbb{F}_2^n$. Let also $\alpha_T \le 1/4$. Then,* normalize(C) *is a circuit computing $f|_R$ such that*

$$\mu(\text{normalize}(C), R) \le \mu(C, R)$$

Proof. Transformations 1–5 do not change the function computed by a circuit and do not affect an underlying rdq-source. Each of them eliminates an internal gate from a circuit, introduces at most four new troubled gates (see Proposition 3.4.3), and does not increase the number of influential input gates. Thus, each application of a transformation rule decreases μ by at least $(1 - 4\alpha_T) \ge 0$. $\square$

We now present the proof of Theorem 3.2.

Proof of Theorem 3.2. Overview. In the proof of the theorem we perform a single step of the amortized complexity analysis of a circuit and an rdq-source. In this step we simplify this pair by substituting a single variable (or more) and/or modifying the rdq source (for example, by adding an equation to it), and it allows us to reduce the complexity of the circuit where the reduction is measured using the measure μ. The circuit simplification is being made using the transformation rules from subsection 3.4 (from simple ones that result from substitutions by constants to the most complicated ones that create cycles). We prove that for every circuit and rdq-source we are able to reduce the measure by the required amount, and it requires considering multiple cases.

Details. Since normalization does not change R and does not increase $\mu(C, R)$, for proving a lower bound on $\mu(C, R)$ one may assume that C is normalized.

Assume that the rdq-source $R \subseteq \mathbb{F}_2^n$ is specified by sets F, L, Q. In what follows, we will further restrict R by decreasing the number of free variables either by one or by two, then we will implement these substitutions in C and normalize C afterwards. Formally, we do it as follows:

- We add an equality or two to R where each equality is of one of the four types stated in Lemma 3.1.1.
- Since the circuit C computes the disperser on the smaller set $R \subseteq \mathbb{F}_2^n$, some gates of C become redundant. (For example, if we add an equality $x_2 \leftarrow x_1 \oplus x_3 \oplus 1$ to R, then a gate computing $x_1 \oplus x_2 \oplus x_3$ is equal to 1 on all points $x \in R$.) We simplify C as follows.
  - Change the operations in the gates fed by the substituted variables or restructure the xor part of the circuit according to Lemma 3.4.2.
  - After then, apply normalizing transformations to eliminate some gates (and disconnect substituted variables).
- We estimate the decrease of μ.

Thus, the gates are eliminated from the circuit by normalizing transformations only. This allows us to keep track of the newly introduced troubled gates via Proposition 3.4.3 that ensures that every gate eliminated by a normalizing transformation introduces at most four new troubled gates. This also shows that normalizing transformations serve two purposes: to assume that we are dealing with a normalized circuit that does not contain obvious inefficiencies and to eliminate gates that become redundant after adding a new equality to R.

Since $s \geq d + 2$, even if we add two more equalities to R, the disperser will not become a constant. This, in particular, implies that if a gate becomes constant, then it is not an output gate and hence feeds at least one other gate. By going through the possible cases we will show that it is always possible to perform one or two consecutive substitutions matching at least one of the following types. By $\Delta\mu$ we denote the decrease of μ after subsequent normalization.

Types of substitutions

1. Perform two consecutive affine substitutions to reduce the number of influential inputs by three. That is, for two different variables $x_j, x_k \in F$, add equalities $x_j = \oplus_{i \in I} x_i \oplus c$ and $x_k = \oplus_{i \in I'} x_i \oplus c'$ to R, then change some wires and gate operations in C, then normalize C. If normalization reduces the number of influential inputs by three, then $\Delta\mu \geq 3\alpha_I$. Per one substitution, this gives $\Delta\mu \geq 1.5\alpha_I$.

2. Perform one affine substitution to reduce the number of influential inputs by two: $\Delta\mu \geq 2\alpha_I$. (This is done similarly to the previous case.)

3. Perform one affine substitution to eliminate four internal gates. That is, for a variable $x_j \in F$, add an equality $x_j = \oplus_{i \in I} x_i \oplus c$ to R, then change some wires and gate operations in C, then normalize C. If normalization removes four internal gates, then $\Delta\mu \geq 4\beta + \alpha_I$.

4. Perform one constant substitution to eliminate three internal gates including at least one troubled gate so that no new troubled gate is introduced. That is, for a variable $x_j \in F$, add an equality

$x_j = c$ to R, then change some wires and gate operations in C, then normalize C. If normalization removes three (internal) gates, one of which is troubled, and *does not introduce new troubled gates*, then $\Delta\mu \geq \alpha_I + 3 + \alpha_T$.

5. Perform one *quadratic* substitution to eliminate five internal gates. That is, for a variable $x_j \in F$, add an equality $x_j = (x_i \oplus a)(x_j \oplus b) \oplus c$ to R, then change some wires and gate operations in C, then normalize C. If normalization removes five gates, then $\Delta\mu \geq 5\beta - \alpha_Q + \alpha_I$ (the α_Q summand corresponds to the newly introduced quadratic equality).

6. Perform two affine substitutions to eliminate five internal gates and replace a quadratic equality by an affine one. That is, for two different variables $x_j, x_k \in F$, add equalities $x_j = \oplus_{i\in I} x_i \oplus c$ and $x_k = \oplus_{i\in I'} x_i \oplus c'$ to R, then change some wires and gate operations in C, then normalize C. If adding these two equalities replaces a quadratic equality in R by an affine one and if normalization eliminates five internal gates, then $\Delta\mu \geq 5\beta + \alpha_Q + 2\alpha_I$. Per one substitution, this gives $\Delta\mu \geq 2.5\beta + \frac{\alpha_Q}{2} + \alpha_I$.

7. Perform one affine substitution to eliminate two internal gates and replace one quadratic equality by an affine one. That is, for a variable $x_j \in F$, add an equality $x_j = \oplus_{i\in I} x_i \oplus c$ to R, then change some wires and gate operations in C, then normalize C. If adding this equality replaces a quadratic equality in R by an affine one and if normalization eliminates two internal gates, then $\Delta\mu \geq 2\beta + \alpha_Q + \alpha_I$.

All substitutions that we perform are of the form such that adding the corresponding equality to the rdq-source results in a new rdq-source (recall Lemma 3.1.1). Note also that δ is defined (recall Equation (3)) as the minimum of the lower bounds on $\Delta\mu$ given in the seven cases above.

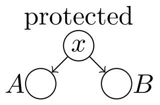
We check all possible cases of (C, R). In every case, we assume that the conditions of the previous cases are not satisfied. We also rely on the specified order of applications of the normalizing transformations where applicable.

Recall that Proposition 3.4.3 guarantees that each of the Transformation rules introduces at most 4 troubled gates while eliminating an internal gate. Thus, $\beta = 1 - 4\alpha_T$ is the minimum guaranteed decrease of the measure per one eliminated gate. Due to our choice of $\beta \geq 0$, if some additional gate disappears during the process, it may introduce new troubled gates but does not increase the measure.

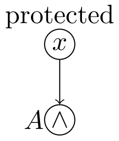
In all the cases below, we use the following observation. Assume that a gate A is fed by gates P and Q such that P is constant on all points of the underlying rdq-source $R \subseteq \mathbb{F}_2^n$. This means that A depends on Q only and computes one of the following four functions: $0, 1, Q, \overline{Q}$. This, in turn, means that A can be eliminated by either Transformation 2 or Transformation 3.

Cases

Case 1. There is a protected (w.r.t. R) variable x that feeds in C either an and-type gate or at least two internal gates. In this case, one can eliminate two gates as follows.



If x feeds two gates A and B, assign 0 to it: add an equality $x = 0$ to R, change operations computed in A and B to passing, and apply Transformation 3 to eliminate these two gates. Since assigning a constant to x removes a quadratic substitution from R (recall that x is protected w.r.t. R), this gives a type 7 substitution.



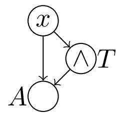
If x feeds an and-type gate A, assign x the constant c that forces A to compute a constant b: add an equality $x = c$ to R; change the operation in A to b and change the operation in its descendant to passing; apply Transformations 2 and 3 to remove these two gates. Again, assigning a constant to x removes a quadratic substitution from R which results in a type 7 substitution.

Case 2. The rdq-source R contains a quadratic equality involving protected variables x and x' such that x is a 0-variable in C (i.e., it feeds no gates). Then, we assign 0 to x': add an equality $x' = 0$ to R, replace the quadratic equality by an affine one, make the corresponding changes to C and normalize it. After this, neither x nor x' are influential, so we have a type 2 substitution.

Note that after this case all protected variables are 1-variables feeding xor gates.

Case 3. There is a variable x feeding an and-type gate T, and $out(x) + out(T) \geq 4$. Let c be the constant such that T becomes a constant b when $x = c$. Then, we add an equality $x = c$ to R, replace x by c in C, and normalize C. Below, we show that normalization eliminates four gates from C, leading to a type 3 substitution.

Note that normalization removes all descendants of x and T. If T and x have no common descendants, this already gives four gates, so assume that they do share descendants and denote by A one of such gates.

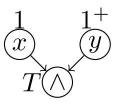


Under $x = c$, A becomes constant, too. Hence, A cannot be an output gate (the dimension of the disperser is at least $d + 2$, hence one cannot trivialize the circuit by a single substitution). Thus, one eliminates T, A, and all descendants of x, T, and A. If there are two such gates, one gets four eliminated gates. If there is a single such gate, it must be a descendant of A and one of x and T. Then, this descendant also turns into a constant, hence it is not an output gate, and hence all its descendants are also eliminated.

Note that after this case all variables feeding and-gates have outdegree one or two.

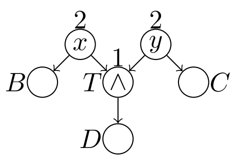
Case 4. There is an and-type gate T fed by two input gates x and y, one of which (say, x) has outdegree 1.

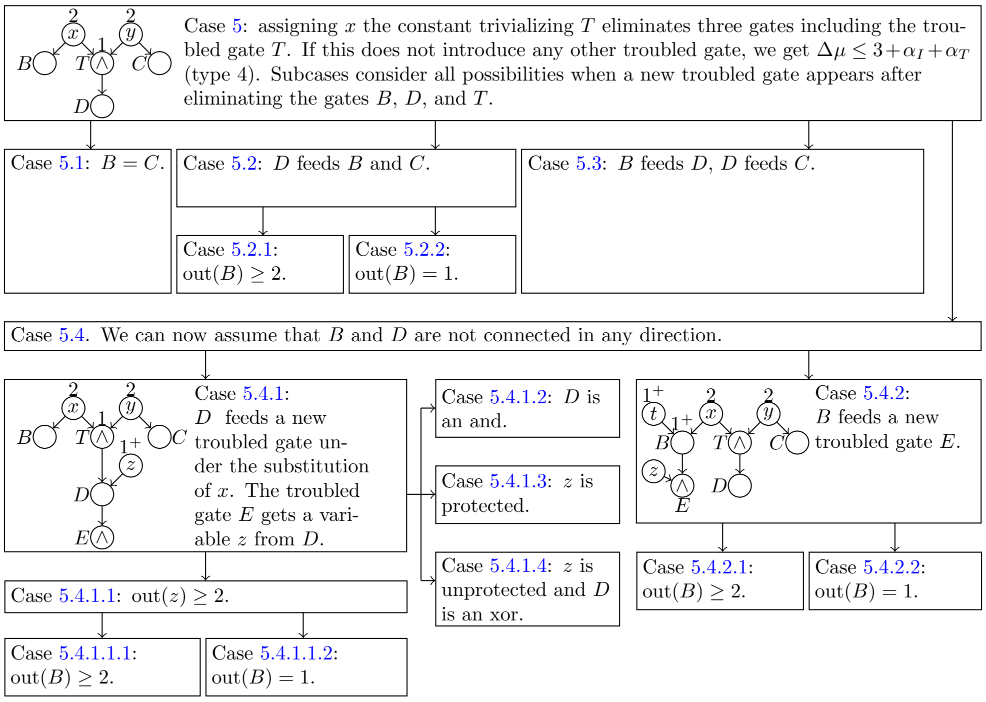
In this and all the subsequent pictures, we show the outdegrees near the gates that are important for the case analysis, similarly to the picture corresponding to the present case.



We substitute y by the constant trivializing T. This removes the dependence on x and y (which are both influential and unprotected), a type 2 substitution.

Case 5. There is an and-type gate T fed by two input gates x and y, and at this point (due to the cases 3 and 4) we have $out(T) = 1$ and $out(x) = out(y) = 2$, that is, T is troubled. We call the gates as described in the following picture:




Case 5: assigning x the constant trivializing T eliminates three gates including the troubled gate T. If this does not introduce any other troubled gate, we get $\Delta\mu \leq 3+\alpha_I+\alpha_T$ (type 4). Subcases consider all possibilities when a new troubled gate appears after eliminating the gates B, D, and T.

Case 5.1: $B = C$.

Case 5.2: D feeds B and C.

Case 5.2.1: $\text{out}(B) \geq 2$.

Case 5.2.2: $\text{out}(B) = 1$.

Case 5.3: B feeds D, D feeds C.

Case 5.4. We can now assume that B and D are not connected in any direction.

Case 5.4.1: D feeds a new troubled gate under the substitution of x. The troubled gate E gets a variable z from D.

Case 5.4.1.1: $\text{out}(z) \geq 2$.

Case 5.4.1.1.1: $\text{out}(B) \geq 2$.

Case 5.4.1.1.2: $\text{out}(B) = 1$.

Case 5.4.1.2: D is an and.

Case 5.4.1.3: z is protected.

Case 5.4.1.4: z is unprotected and D is an xor.

Case 5.4.2: B feeds a new troubled gate E.

Case 5.4.2.1: $\text{out}(B) \geq 2$.

Case 5.4.2.2: $\text{out}(B) = 1$.

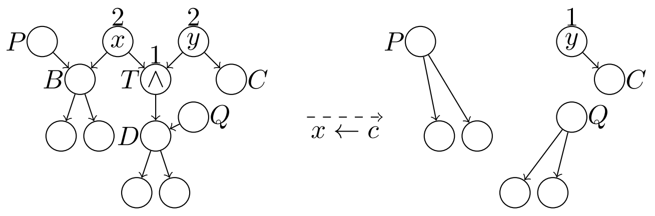
Figure 5: Subcases of Case 5.

Since the circuit is normalized, $B \neq D$ and $C \neq D$ (Transformation 4). One eliminates B, T, D by substituting the constant to x that trivializes T. If, in addition to the three gates, one more gate is eliminated by normalization, we are done (substitution of type 3). Otherwise, we have just three gates, but the troubled gate T is eliminated. If no new troubled gate is introduced during normalization, it makes a substitution of type 4. Likewise, if this is the case for a substitution to y, we are done.

In the remaining subcases of Case 5 we will be dealing with the situation where only three gates are eliminated while one or more troubled gates are introduced (see Figure 5).

Note that a new troubled gate is introduced only if something has happened around some and-type gate E, which becomes a new troubled gate. Consider the case we are substituting x (not y). Whatever has happened, it is due to two gates, B and D, that became passing (if some of them became trivial, then one more gate would be removed).

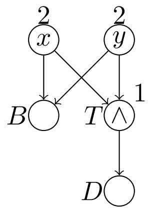
Denote by P and Q the two gates that feed B and D, respectively. Then, after the normalization the circuit looks as follows.



The gates B and D become passing, so after the normalization, P feeds the successors of B and Q feeds the successors of D. Thus, the out-degree of P and Q cannot decrease and hence these two gates cannot become troubled after the normalization. Thus, the only gates that could potentially become troubled are the successors of B and D.

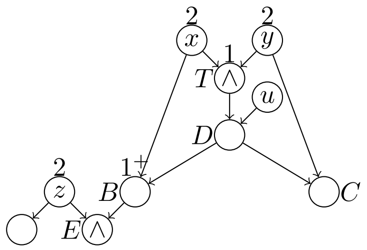
The plan for the following cases is to exploit the local topology, that is, possible connections between B, D, and C. First we consider "degenerate" cases where these gates are locally connected beyond what is shown in the figure in Case 5. After this, we continue to the more general case.

Case 5.1. $B = C$.



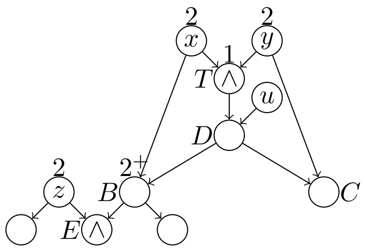
We show that it is possible to trivialize both T and B by a single affine substitution, giving a substitution of type 2 (x and y are unprotected so the number of influential variables drops by 2). Assume that T computes the operation $(x \oplus a)(y \oplus b) \oplus c$ where $a, b, c \in \{0, 1\}$. If B is an $\oplus$-type gate, it computes $x \oplus y \oplus d$ for $d \in \{0, 1\}$. Then, under the substitution $x \leftarrow y \oplus a \oplus b \oplus 1$ both T and B trivialize (the gate T evaluates to c, since, under the considered substitution, $x \oplus a = y \oplus b \oplus 1$ and hence $(x \oplus a)(y \oplus b) = 0$). Assume now that B is an $\wedge$-type gate computing $(x \oplus a')(y \oplus b') \oplus c'$ for $a', b', c' \in \{0, 1\}$. If either $a = a'$ or $b = b'$, then the two gates are trivialized by either $x \leftarrow a$ or $y \leftarrow b'$. Hence, the only remaining case is when $a' = a \oplus 1$ and $b' = b \oplus 1$. In this case, a substitution $x \leftarrow y \oplus a \oplus b \oplus 1$ trivializes T and B: B computes $(x \oplus a \oplus 1)(y \oplus b \oplus 1) \oplus c'$ and this product is equal to 0 under the substitution.

Case 5.2. D feeds both B and C.

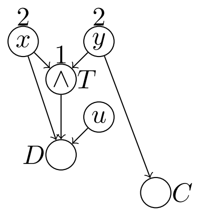


In this case, a new troubled gate may emerge (after substituting x to trivialize T and normalizing) only because D is fed by a variable u, and it is passed to some and-type gate E. Note that $out(D) \le 2$, because otherwise u would become a 3-variable and E would not become troubled. Therefore, u cannot be passed by D to E directly, it is passed via B.

Case 5.2.1. $out(B) \ge 2$.

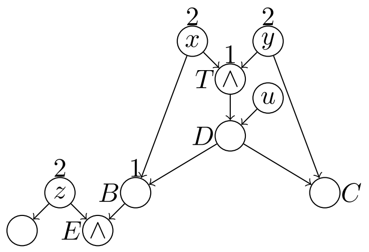


Even if $out(u) = 1$, it must be that $C = E$ or that B feeds C, because otherwise u would become a 3-variable after substituting x. Below we show that neither is possible.



If $C = E$, we have that $B = D$ and $y = z$ (E is fed by B, whereas C is fed by y and D; hence, if $C = E$, then the in-degree of C is more than two unless $B = D$), contradicting the assumption that $D \neq B$ (from Case 5). If B feeds C, we have that $B = D$, which contradicts the assumption that $B \neq D$ from Case 5.

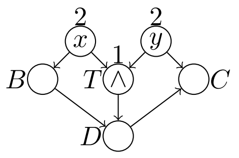
Case 5.2.2. $out(B) = 1$.



We can substitute constants for z, to make B a 0-gate, and for y, to trivialize T. This way x ceases to be influential, and we have $\Delta\mu \geq 3\alpha_I$ for two substitutions (type 1).

Note that after this case we can assume that D does not feed B. If it does, we switch the roles of the variables x and y.

Case 5.3. B feeds D and D feeds C.

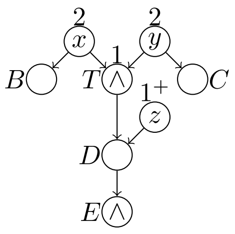


Substituting y to trivialize T removes T, D, and C. Now we show that this substitution introduces *no new troubled gates*, which contradicts our assumption about new troubled gates. The gates C and D are passing the internal gate B. Thus, the gate that used to be fed by C is now fed by B, therefore, locally nothing changed for this gate. The only gate that now locally looks differently is the gate B, but it is now fed by the variable x of degree 1, and, therefore, is not a troubled gate.

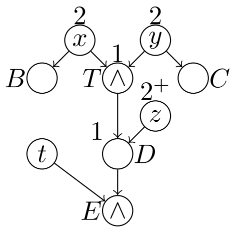
Case 5.4. We can now assume that there is no wire between B and D.

Indeed, if B feeds D, we can switch the roles of x and y unless C feeds D (impossible, because then D has three inputs: T, B, and C) or unless we switched x and y before (that is, D feeds C, Case 5.3).

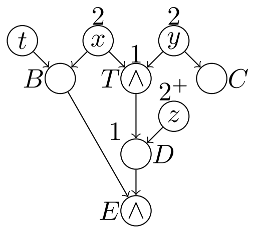
Case 5.4.1. D feeds a new troubled gate under the substitution of x. The troubled gate E gets some variable z from D (directly, as D and B are not connected).



Case 5.4.1.1. $out(z) \geq 2$. Then, $out(D) = 1$ and E is fed by another variable t either directly or via B.

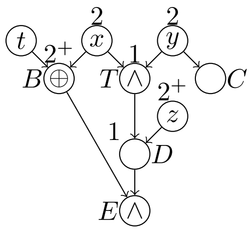


If E is fed by another variable t, we substitute t to trivialize E. Then, E is eliminated by Transformation 2, its descendant becomes either passing or constant and is eliminated by Transformation 3 or 2. Also, the gate D becomes a 0-gate and is eliminated by Transformation 1. Then, T is eliminated by Transformation 1, giving a type 3 substitution.



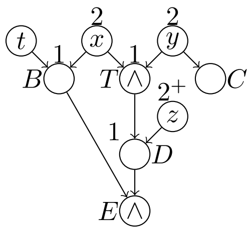
In the latter case when B is fed by t so that after normalizing, E is fed by t, we consider two subcases.

Case 5.4.1.1.1. $out(B) \geq 2$.



In this case, B is an xor-type gate (see Case 3), and by substituting $x = t \oplus c$ for the appropriate constant c, we can make B a constant trivializing E and eliminate two more descendants of B and E, a type 3 substitution. (If there is just a single gate fed by B and E, then it becomes constant under the substitution, so at least one more descendant of this gates is eliminated.)

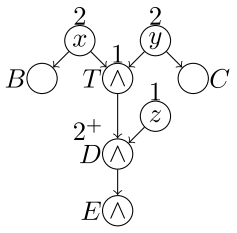
Case 5.4.1.1.2. $out(B) = 1$.



We set z and y to constants trivializing T and E, respectively. Then B becomes a 0-gate and is eliminated, which means that x becomes a 0-variable. We then get a substitution of type 1.

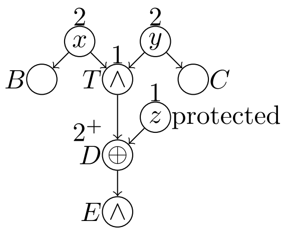
We can now assume that $out(z) = 1$ and thus $out(D) \geq 2$, because z must get outdegree two in order to feed the new troubled gate.

Case 5.4.1.2. D is an and-type gate.



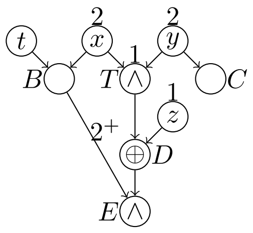
Substituting z by the appropriate constant trivializes D and eliminates both gates that it feeds (recall that $out(D) \geq 2$); also T becomes a 0-gate, a type 3 substitution.

Case 5.4.1.3. z is protected (recall that a variable is called protected if it occurs in the right-hand side of a quadratic equality defining the current rdq-source R).



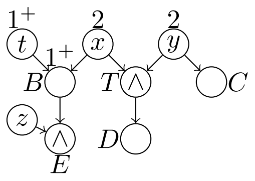
We first substitute x by the constant trivializing x. After the normalization, the gates B, T, and D are eliminated and E is fed by t and z. We then substitute z by the constant trivializing E. Then, the normalization eliminates E and its descendants (note that E cannot feed any of B, D, T as this would create a cycle). Thus, we eliminate five gates and one quadratic equality, a type 6 substitution.

Case 5.4.1.4. Since we can now assume that z is unprotected and D is an xor-type gate, D computes the function $(x \oplus a)(y \oplus b) \oplus z \oplus c$ (for $a, b, c \in \{0, 1\}$).

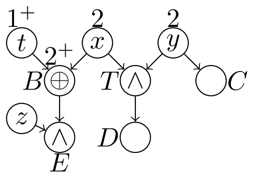


Since E is an $\wedge$-type gate, it computes a function $(B \oplus a')(D \oplus b') \oplus c'$ (for $a', b', c' \in \{0, 1\}$). We make a *quadratic* substitution $z \leftarrow (x \oplus a)(y \oplus b) \oplus c \oplus b'$. To do this, we add the corresponding quadratic equality and mark x and y as protected. (Note that $out(z) = 1$, so we do not need to replace other occurrences of z with its new value.) Under this substitution, the gates D and E compute the constants b' and c', respectively, and are eliminated by Transformation 2. Then, T becomes a 0-gate and is eliminated by Transformation 1. Since D and E become constant, all their descendants are also eliminated. If there are at least two such gates, then the total number of eliminated gates is five. If D and E share a descendant F, then F becomes constant and all descendants of F are also eliminated. Thus, in any case five gates are eliminated. Taking into account the penalty for introducing a quadratic equality, we get a substitution of type 5.

Case 5.4.2. Since D does not feed a new troubled gate, B does, and B is fed directly by a variable t (since B and D are not connected). The new troubled gate E must be also fed directly by a variable z (because D does not feed it).

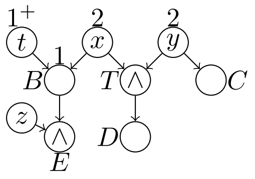


Case 5.4.2.1. $out(B) \geq 2$. In this case, B is an xor-type gate, see Case 3).



Let c be a constant such that E is trivialized under $x \leftarrow t \oplus c$. We make a substitution $x \leftarrow t \oplus c$ using Proposition 3.4.2. This does not introduce new troubled gates, since $out(x) \neq 1$. Normalizing eliminates B, E, and their descendants, a type 3 substitution. (If B and E share a descendant F, then F becomes constant and all descendants of F are also eliminated.)

Case 5.4.2.2. $out(B) = 1$.



In this case, we can set z and y to constants trivializing T and E, respectively. Then B becomes a 0-gate and is eliminated, which means that x becomes a 0-variable. We then get a substitution of type 1.

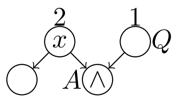
Starting from the next case we will consider a topologically minimal and-type gate and call it A for the remaining part of the proof. Here A is topologically minimal if it cannot be reached from another and-type gate via a directed path. (Note that there are no cycles containing and-type gates in a fair semicircuit. Thus, it is always possible to find a topologically minimal and-type gate.)

Note that the circuit C must contain at least one and-type gate. Indeed, otherwise C would compute an affine function $\bigoplus_{i \in I} x_i \oplus c$ on the current rdq-source R of dimension $s \geq d+2$. But then, C would compute the constant on the following rdq-source $R' \subseteq \mathbb{F}_2^n$ of dimension at least $d+1$:

$$R' = \left\{ x \in R \colon \bigoplus_{i \in I} x_i = 0 \right\} .$$

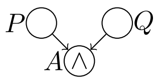
The minimality implies that both inputs of A are computed by fair cyclic xor-circuits (note that a subcircuit of a fair circuit is fair, because it corresponds to a submatrix of a full-rank matrix); in particular, they can be input gates.

Case 6. One input of A is an input gate x of outdegree 2 while the other one is an internal gate Q of outdegree 1.

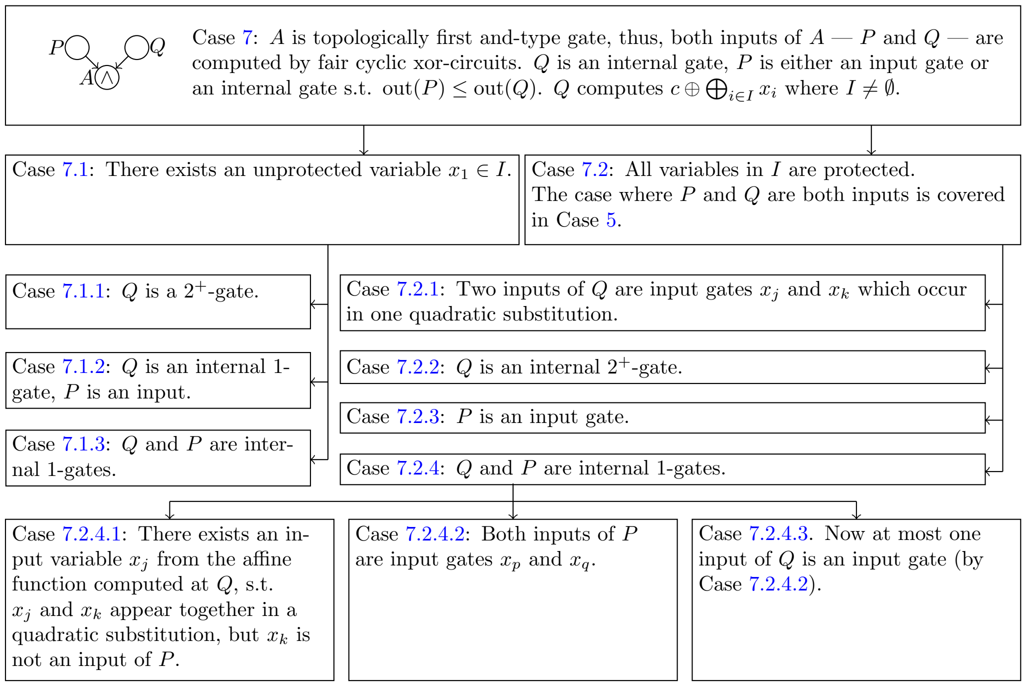


Recall that x is unprotected due to Case 1, and x cannot feed Q because of Transformation 4. Substituting x by the constant trivializing A eliminates the two successors of x, all the successors of A, and makes Q a 0-gate which is then eliminated by Transformation 1. A type 3 substitution. (As usual, if the only successor of A coincides with the other successor of x then this gate becomes constant so its successors are also eliminated. That is, in any case at least four gates are eliminated.)

Case 7. One input to A is an internal gate Q. Denote the other input by P. If P is also an internal gate and has outdegree larger than Q we switch the roles of P and Q.



In this case we will try to substitute a value to Q in order to trivialize A (see Figure 6). Q is a gate computed by a fair xor-circuit, so it computes an affine function $c \oplus \bigoplus_{i\in I} x_i$. Note that $I \neq \emptyset$ because of Transformation 2. We use the xor-reconstruction procedure described in Lemma 3.4.2. In order to perform it, we need at least one unprotected variable x_i with $i \in I$.


Case 7: A is topologically first and-type gate, thus, both inputs of A — P and Q — are computed by fair cyclic xor-circuits. Q is an internal gate, P is either an input gate or an internal gate s.t. $\text{out}(P) \le \text{out}(Q)$. Q computes $c \oplus \bigoplus_{i\in I} x_i$ where $I \neq \emptyset$.

Case 7.1: There exists an unprotected variable $x_1 \in I$.

Case 7.2: All variables in I are protected. The case where P and Q are both inputs is covered in Case 5.

Case 7.1.1: Q is a 2^+-gate.

Case 7.1.2: Q is an internal 1-gate, P is an input.

Case 7.1.3: Q and P are internal 1-gates.

Case 7.2.1: Two inputs of Q are input gates x_j and x_k which occur in one quadratic substitution.

Case 7.2.2: Q is an internal 2^+-gate.

Case 7.2.3: P is an input gate.

Case 7.2.4: Q and P are internal 1-gates.

Case 7.2.4.1: There exists an input variable x_j from the affine function computed at Q, s.t. x_j and x_k appear together in a quadratic substitution, but x_k is not an input of P.

Case 7.2.4.2: Both inputs of P are input gates x_p and x_q.

Case 7.2.4.3. Now at most one input of Q is an input gate (by Case 7.2.4.2).

Figure 6: Subcases of Case 7.

Case 7.1. Such a variable x_1 exists.

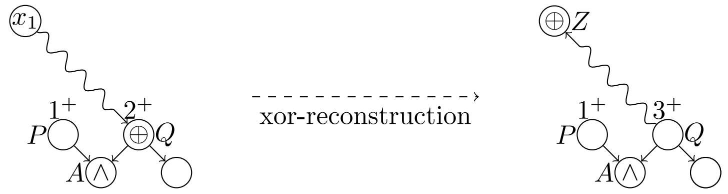
We then add the equality $x_1 = b \oplus c \oplus \bigoplus_{i\in I\setminus\{1\}} x_i$ to the rdq-source R for the appropriate constant b (so that Q on the updated R computes the constant trivializing A). We could now simply replace the operation in Q by this constant (since the just updated circuit computes correctly the disperser on the just updated R). However, we need to eliminate the just substituted variable

x_1 from the circuit. To do this, we perform the reconstruction described in Lemma 3.4.2. Note that it only changes the in- and out-degrees of x_1 (replacing it by a new internal gate Z) and Q. No new troubled gates are introduced (see Corollary 3.4.1), and the subsequent application of Transformation 2 to Q removes Q.

Moreover, normalizing transformations remove all descendants of Q, all descendants of A, and, in the case $out(P) = 1$, Transformation 1 removes P if it is an internal gate, or P becomes a 0-variable, if it was a variable. It remains to compute the decrease of the measure.

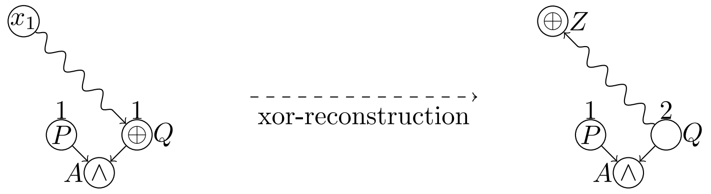
Below we go through several subcases depending on the types and out-degrees of the gates P and Q.

Case 7.1.1. Q is a 2^+-gate. We recall the general picture of xor-reconstruction.



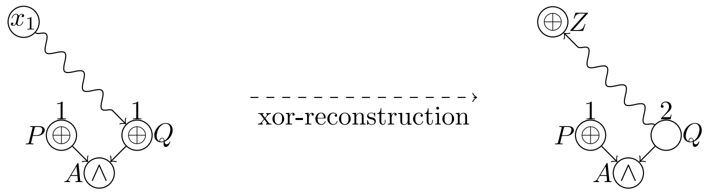
After the reconstruction, there are at least three descendants of Q and at least one descendant of A, a type 3 substitution. (If a descendant of A is also a descendant of Q, then it becomes constant and hence all its descendants are also eliminated.)

Case 7.1.2. Q is an internal 1-gate and P is an input gate. Then P has outdegree 1 and is unprotected (see Cases 6, 1).



Note that $P \neq x_1$ since the only outgoing edge of P goes to an and-type gate. Thus, after the xor-reconstruction and normalization (that eliminates the gate A that is trivialized), the circuit becomes independent of both x_1 and P, giving a type 2 substitution.

Case 7.1.3. Q is an internal 1-gate and P is an internal gate. Then P is a 1-gate (if the outdegree of P were larger we would switch the roles of P and Q).



After the substitution, we remove two successors of Q, at least one successor of A, and make P a 0-gate, giving a type 3 substitution. Note that P cannot be a successor of Q because of Transformation 4. If a descendant of A is also a descendant of Q, then it becomes constant and hence all its descendants are also eliminated.

Case 7.2. All variables in the affine function computed by Q are protected.

Case 7.2.1. Both inputs to Q are variables, say x_j and x_k, and they occur in the same quadratic equality $w = (x_j \oplus c)(x_k \oplus c') \oplus c''$ in R. Perform a substitution $x_j \leftarrow x_k \oplus c'''$ (using Proposition 3.4.2) in order to trivialize the gate A. It does not introduce new troubled gates as x_j is not a 1-gate feeding an and-type gate. It eliminates the quadratic equality from R (and does not harm other quadratic equalities, because x_j and x_k could not occur in them) and eliminates Q, A, its descendant (and more, but we do not need it) from C, which makes $\Delta\mu \geq 3\beta + \alpha_Q + \alpha_I$, a type 7 substitution.

Case 7.2.2. Q is a 2^+-gate. Take any $j \in I$. Assume that x_j occurs in a quadratic equality $x_p = (x_j \oplus a)(x_k \oplus b) \oplus c$ in R. Recall that at this point, all protected variables are 1-variables feeding xor-gates (see Cases 1 and 2). Make a substitution $x_k \leftarrow d$ and normalize the circuit. This eliminates the successor of x_k from C, eliminates the quadratic equality from R, and makes x_j unprotected. If at least two gates are removed during normalization, then we get $\Delta\mu \geq 2\beta + \alpha_Q + \alpha_I$, a type 7 substitution. In what follows, we assume that the only gate removed during normalization after the substitution $x_k \leftarrow d$ is the successor of x_k.

If the gate Q is not fed by x_k, then it has outdegree at least 2 after the substitution $x_k \leftarrow d$ and normalizing the descendants of x_k. If the gate Q is fed by x_k, then its second input must be an internal xor-gate Q' (if it were an input gate it would be a variable x_j but then we would fall into Case 7.2.1). Then, after substituting $x_k \leftarrow d$ and normalizing Q the gate Q' feeds A and has outdegree at least 2. We denote Q' by Q in this case.

Hence in any case, in the circuit normalized after the substitution $x_k \leftarrow d$, the gate A is fed by the 2^+-gate Q that computes an affine function of variables containing an unprotected variable x_j. We then make Q constant trivializing A by the appropriate affine substitution to x_j. Like in Case 7.1.1, this eliminates four gates: after the xor-reconstruction, Q is a 3^+-gates, the normalization removes all its descendants (including A); since A becomes constant, all its descendants are also eliminated; if a descendant of A is also a descendant of Q, then it becomes constant and hence all its descendants are also eliminated. Together with the substitution $x_k \leftarrow d$, it gives $\Delta\mu \geq 5\beta + \alpha_Q + 2\alpha_I$, a type 6 substitution.

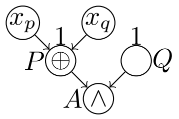
Hence in what follows we assume that $out(Q) = 1$. *Therefore P is either a variable or an internal xor-type* 1*-gate.*

Case 7.2.3. P is an input gate. Then P has outdegree 1 and is unprotected (see Cases 6, 1). Take any $j \in I$ and assume that x_j appears with x_k in a quadratic equality in R. We first substitute $x_k \leftarrow d$ and normalize the circuit. After this, the second input of A still computes a linear function that depends on x_j which is now unprotected. We make an affine substitution to x_j trivializing A. This makes P a 0-variable, a type 1 substitution.

Case 7.2.4. P is an internal xor-type 1-gate. If P computes an affine function of variables at least one of which is unprotected, we are in Case 7.1.3 with P and Q exchanged. So, in what follows we assume that both P and Q compute affine functions of protected variables.

Case 7.2.4.1. One of P and Q (say, Q) computes an affine function of variables one of which (call it x_j) has a couple x_k that does not feed P.[2] We substitute x_k by a constant and normalize the descendant of x_k. Normalization eliminates one xor-gate fed by x_k and makes x_j unprotected. Note that at this point P is still an xor-type 1-gate. We then trivialize A by substituting x_j by an affine function. Similarly to Case 7.1.3, this eliminates four gates and gives, for two substitutions, $\Delta\mu \geq 5\beta + \alpha_Q + 2\alpha_I$. A type 6 substitution.

Case 7.2.4.2. Both inputs to P or Q (say, P) are (protected) variables x_p and x_q. Let x_j be a (protected) variable from the affine function computed at Q and let x_k be its couple. Note that by Case 1, $x_j \neq x_p, x_q$. By Case 7.2.4.1, we may assume that $x_k = x_p$ or $x_k = x_q$. For concreteness, let us assume that $x_k = x_p$.



We substitute x_p by a constant to eliminate a quadratic equality containing x_p. The gate P then computes x_q or its negation. We substitute x_q by a constant that trivializes

[2]We say that x_k is a couple of x_j if they appear in the right-hand side of the same quadratic substitution.

A. This way, we eliminate gates A, P, Q, and a successor of A. All these four gates are different: P and Q are predecessors of A and hence cannot be successors of A. Also, the number of influential variables reduces by two (x_p and x_q) and the number of quadratic equalities reduces by two (the quadratic equalities containing x_p and x_q are distinct, because the couple of x_p is $x_j \neq x_q$). These two substitutions decrease the measure by at least $\Delta\mu \geq 4\beta + 2\alpha_Q + 2\alpha_I$, which corresponds to two substitutions of type 7.

Case 7.2.4.3. By case 7.2.4.2, P and Q are fed by at most one input gate. If P (or Q) computes an affine function which depends on two or more variables, then at least one of those variables does not feed Q (or P), which is covered by Case 7.2.4.1. Therefore, the only remaining case is the following. Since P and Q, and gates that feed them all compute nontrivial functions (because of Transformation 2), P computes an affine function of a single variable x_p, Q computes an affine function of a single variable x_q, the variables x_p and x_q appear together in a quadratic equality. Moreover x_p feeds Q while x_q feeds P: if, say, x_p does not feed Q, then we are in Case 7.2.4.1. But this is just impossible. Indeed, since x_p is a protected variable it only feeds Q (by Case 1). As P computes an affine function on x_p, Lemma 3.4.1 guarantees that there is a path from x_p to P. But this path must go through Q and A leading to a cycle that goes through an and-type gate A.

□

4 Lower bound $3.11n$ for quadratic dispersers

This section is devoted to the proof of the following theorem.

Theorem 1.2. *The circuit size of an* $(n, 1.83n, 2^{o(n)})$*-quadratic disperser is at least* $3.11n$.

4.1 Xor-inputs circuits

By an *xor-inputs* circuit we mean a circuit whose inputs may be labeled not only by input variables but also by sums of variables. We note that one can get rid of xor-type top-gates in an xor-inputs circuit (a top-gate is a gate fed by two input-gates). To this end one can repeatedly replace xor-gates that depend on two inputs by new inputs (see Figure 7). Moreover, such a transformation does not increase the total number of inputs and internal gates. ***In this section we abuse the notation by using the word "circuit" to mean an xor-inputs circuit.***

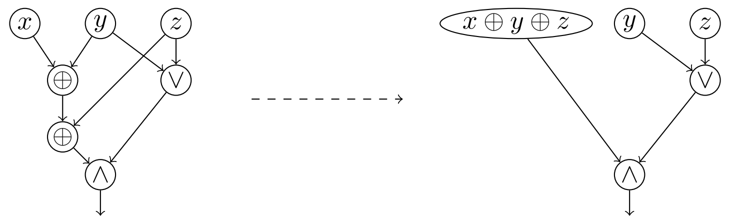


Figure 7: An example of a transformation from a regular circuit to an xor-inputs circuit.

In this section, we use a circuit complexity measure: $\mu(C) = G(C) + \alpha \cdot I(C)$ where $0 < \alpha \leq 1$ is a constant to be determined later. Clearly, the transformation above does not increase $\mu(C)$.

4.2 Lower Bound

Theorem 1.2 follows from the following more general result.

Theorem 4.1. *Let $0 < \alpha \le 1$ and $0 < \beta$ be constants satisfying*

$$2^{-\frac{2+\alpha}{\beta}} + 2^{-\frac{4+2\alpha}{\beta}} \le 1, \tag{4}$$
$$2^{-\frac{2}{\beta}} + 2^{-\frac{5+2\alpha}{\beta}} \le 1, \tag{5}$$
$$2^{-\frac{3+3\alpha}{\beta}} + 2^{-\frac{2+2\alpha}{\beta}} \le 1, \tag{6}$$
$$2^{-\frac{3}{\beta}} + 2^{-\frac{4+\alpha}{\beta}} \le 1, \tag{7}$$

and let $f \in B_n$ be an (n, k, s)-quadratic disperser. Then

$$C(f) \ge \min\{\beta n - \beta \log_2 s - \beta, 2k\} - \alpha n\,.$$

The "magic" numbers α and β control the decrease in the circuit complexity and the decrease in the variety size: informally, the numerator (e.g., $2 + \alpha$) corresponds to the decrease in the complexity measure (which takes into account the number of internal gates and the number of variables (input gates)) for a particular substitution and the exponent (for example, $2^{-\frac{2+\alpha}{\beta}}$) upper bounds the decrease in the variety size after this substitution.

Proof of theorem 1.2. It is not difficult to check that $\alpha = 0.535$ and $\beta = 3.6513$ satisfy the conditions of theorem 4.1. Since f is a quadratic disperser with parameters $k = 1.83n$ and $s = 2^{o(n)}$, theorem 4.1 gives the following lower bound on $C(f)$:

$$\min\{3.6513n - o(n), 3.66n\} - 0.535n = 3.1163n - o(n) > 3.11n\,.$$

□

We need the following technical lemma.

Lemma 4.2.1. *Let $0 < \alpha \le 1$ and $0 < \beta$ be constants satisfying inequalities eq. (7) and eq. (4). Then*

$$2 \cdot 2^{-\frac{4}{\beta}} \le 1, \tag{8}$$
$$2^{-\frac{3+\alpha}{\beta}} + 2^{-\frac{3+2\alpha}{\beta}} \le 1. \tag{9}$$

Proof. Since $2 \le x + \frac{1}{x}$ for positive x,

$$2^{-\frac{4}{\beta}} + 2^{-\frac{4}{\beta}} \le 2^{-\frac{4}{\beta}}(2^{\frac{1}{\beta}} + 2^{-\frac{1}{\beta}}) = 2^{-\frac{3}{\beta}} + 2^{-\frac{5}{\beta}} \le 2^{-\frac{3}{\beta}} + 2^{-\frac{4+\alpha}{\beta}} \le 1\,.$$

In order to prove the inequality eq. (9), we use Heinz's inequality [22]:

$$x^{1-t}y^t + x^t y^{1-t} \le x + y \text{ for } x, y > 0, 0 \le t \le 1.$$

Let us take $x = 2^{-\frac{2+\alpha}{\beta}}, y = 2^{-\frac{4+2\alpha}{\beta}}, t = \frac{1}{2+\alpha}$:

$$2^{-\frac{3+\alpha}{\beta}} + 2^{-\frac{3+2\alpha}{\beta}} = x^{1-t}y^t + x^t y^{1-t} \le x + y = 2^{-\frac{2+\alpha}{\beta}} + 2^{-\frac{4+2\alpha}{\beta}} \le 1.$$

□

Theorem 4.1 follows immediately from the following lemma with $S = \mathbb{F}_2^n$ that is an $(n, 0)$-quadratic variety. (Recall that an (n, k)-quadratic variety is a set $S \subseteq \mathbb{F}_2^n$ such that there exists k polynomials $p_1, \ldots, p_k$ of degree at most 2 such that $S = \{x \in \mathbb{F}_2^n : p_1(x) = 0, \ldots, p_k(x) = 0\}$. See Definition 2.2.2.)

Lemma 4.2.2. *Let $f \in B_n$ be an (n, k, s)-quadratic disperser, $S \subseteq \mathbb{F}_2^n$ be an (n, t)-quadratic variety, $0 < \alpha \le 1, 0 < \beta$ be constants satisfying inequalities eq. (4), eq. (5), eq. (6), eq. (7), C be an xor-inputs circuit that computes f on S. Then*

$$\mu(C) \ge \min\{\beta(\log_2 |S| - \log_2 s - 1), 2(k - t)\}\,.$$

Proof. The proof goes by induction on $|S|$. The base case $|S| \le 2s$ is trivially true. For the induction step, assume that $|S| > 2s$.

To prove the induction step we proceed as follows. If $t \ge k$ then the right-hand side is non-positive, so assume that $t < k$. Assume that C is optimal with respect to μ (that is, C has the minimal value of μ among all circuits computing f on S). We find a gate G in C that computes a polynomial g of degree at most 2 and consider two $(n, t+1)$-quadratic varieties of S: $S_0 = \{x \in S \colon g(x) = 0\}$ and $S_1 = \{x \in S \colon g(x) = 1\}$. Let $|S_0| = p_0|S|$ and $|S_1| = p_1|S|$ where $0 < p_0, p_1 < 1$ and $p_0 + p_1 = 1$ (note that $p_i = 0$ or $p_i = 1$ would mean that G computes a constant on S contradicting the fact that C is optimal). By eliminating from the circuit C all the gates that are either constant or depend on just one of its inputs on S_i, one gets a circuit C_i that computes f on S_i. Assume that $\mu(C) - \mu(C_i) \ge \Delta_i$. Then, by the induction hypothesis,

$$\begin{aligned}\mu(C) \ge \mu(C_i) + \Delta_i \ge\\ &\min\{\beta(\log_2|S_i| - \log_2 s - 1), 2(k - (t+1))\} + \Delta_i =\\ &\quad \min\{\beta(\log_2|S| - \log_2 s - 1) + (\Delta_i + \beta\log_2 p_i), 2(k-t) + (\Delta_i - 2)\}\,.\end{aligned}$$

Hence, if $\Delta_i \ge -\beta\log_2 p_i$ and $\Delta_i \ge 2$ for either $i = 0$ or $i = 1$ then the required inequality follows by the induction hypothesis. The inequality $\Delta_i \ge -\beta\log_2 p_i$ is true whenever $p_i \ge 2^{-\frac{\Delta_i}{\beta}}$. Since we want this inequality to hold for at least one of $i = 0$ and $i = 1$ and since $p_0 + p_1 = 1$ we conclude that for the induction step to go through it suffices to have

$$2^{-\frac{\Delta_0}{\beta}} + 2^{-\frac{\Delta_1}{\beta}} \le 1 \text{ and } \Delta_0, \Delta_1 \ge 2\,. \tag{10}$$

By going through a few cases we show that we can always find a gate G such that the corresponding Δ_0 and Δ_1 satisfy the inequalities eq. (10). (In order to do this, we in particular use the fact that α and β satisfy the inequalities eq. (4)–eq. (9).)

We start by showing that the circuit C must be non-empty. Indeed, if C is empty then it computes a linear function l. Hence f is constant on both $S_0 = \{x \in S \colon l(x) = 0\}$ and $S_1 = \{x \in S \colon l(x) = 1\}$. However $\max\{|S_0|, |S_1|\} \ge |S|/2 > s$ which contradicts the fact that f is an (n, k, s)-quadratic disperser.

In the case analysis below, we find a gate G that computes a polynomial p of degree at most 2 and make it equal to constant c. To do this, we restrict the current set S to $S_c = \{x \in S \colon p(x) = c\}$. Then, for every gate H of C, if it is constant on S_c, we replace its operation by this constant. After this, we normalize the circuit using Transformations 1–3 (see Subsection 3.4.2; it is easy to see that this can only decrease $\mu(C)$).

The gate G in this case cannot be an output gate as otherwise the whole circuit would be constant on both subsets $\{x \in S \colon p(x) = 0\}$ and $\{x \in S \colon p(x) = 1\}$ one of which has size at least $s/2$, contradicting to the disperser property. Hence, G has at least one successor and it is also eliminated.

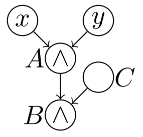
Let A be an and-gate with the maximal number of and-gates on a way to the output of C. That is, for each and-gate we consider all directed paths from this gate to the output gate and select a path with the maximal number of and-gates on it; then we choose an and-gate for which this number is maximal over all and-gates. Since C is an xor-inputs circuit, we may assume that A is a top-gate, that is, it is fed by inputs. Denote by x and y the input gates that feed A.

Case 1. $\mathrm{out}(x) = \mathrm{out}(y) = 1$.

Case 1.1. $\mathrm{out}(A) = 1$ and A feeds an and-gate B.

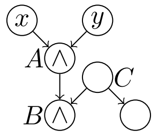
Let C be the other input of B (it might be an input as well as a non-input gate).

Case 1.1.1. $\mathrm{out}(C) = 1$.



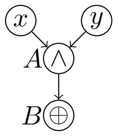
We make A constant. Then the gate B is eliminated. Moreover, either $A = 0$ or $A = 1$ trivializes the gate B so all its successors and the gate C are also eliminated (since C is only used to compute B, but B now computes a constant). In both cases x and y are not needed anymore (as the only gate A that was fed by both these inputs is now constant). So, we get $\{\Delta_0, \Delta_1\} = \{2 + 2\alpha, 3 + 3\alpha\}$ if C is an input gate (or $\{2 + 2\alpha, 4 + 2\alpha\}$ if it is not, but this is even better as $\alpha \leq 1$, which we use in the rest of the analysis without further mentioning it). The required inequalities eq. (10) follow from eq. (6).

Case 1.1.2. $\mathrm{out}(C) \geq 2$.



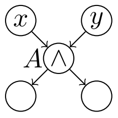
Because of the choice of A, the gate C computes a polynomial of degree at most 2. We make C constant. In both cases we eliminate two successors of C and C itself. This reduces the measure by at least $2 + \alpha$. In one of the cases B is trivialized which causes the removal of the successors of B, the gate A, and inputs x and y. Hence we get $\{\Delta_0, \Delta_1\} = \{2 + \alpha, 4 + 3\alpha\}$ in this case. These Δ_0, Δ_1 satisfy the inequalities eq. (10) because of eq. (4).

Case 1.2. $\mathrm{out}(A) = 1$ and A feeds an xor-gate B.



Since A was chosen as an and-gate with the maximal number of and-gates to the output, the other input of B computes a polynomial of degree at most 2. Hence B itself computes a polynomial of degree at most 2. We make B constant. This eliminates B and its successors. The gate A and its inputs x and y are also not needed. Hence $\Delta_0 = \Delta_1 = 3 + 2\alpha$. The inequalities eq. (10) are satisfied due to eq. (9).

Case 1.3. $\mathrm{out}(A) \geq 2$.



Just by making the gate A constant we get $\Delta_0 = \Delta_1 = 3 + 2\alpha$ since A and all its successors (at least two gates) are eliminated. Similarly to the previous case, the inequality eq. (9) imply that eq. (10) holds.

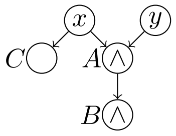
Case 2. $\max\{out(x), out(y)\} \geq 2$. Say, $\mathrm{out}(x) \geq 2$.

Case 2.1. $\mathrm{out}(A) = 1$ and A feeds an and-gate B.

We make A constant. Assume that A computes $(x \oplus c_1)(y \oplus c_2) \oplus c$. Then A can only be equal to $c \oplus 1$ if $x = c_1 \oplus 1$ and $y = c_2 \oplus 1$. That is, when A is equal to $c \oplus 1$ not only its successor is eliminated but also all successors of x and y. In both cases the gate B is eliminated, but in one of them it is trivialized and so all its successors are also eliminated.

Denote by C another gate fed by x. Note that $B \neq C$ (otherwise the circuit would not be optimal).

Case 2.1.1. $\mathrm{out}(y) = 1$.



Case 2.1.1.1. B is trivialized when $A = c$.

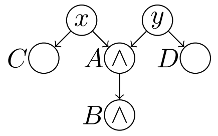
If $A = c$ we eliminate A, B, the successors of B, and y. If $A = c \oplus 1$ we eliminate A, B, C, x, and y. Hence $\{\Delta_0, \Delta_1\} = \{3 + \alpha, 3 + 2\alpha\}$. The inequality eq. (9) guarantees that eq. (10) holds.

Case 2.1.1.2. B is trivialized when $A = c \oplus 1$.

If $A = c$ we eliminate A, B, and y. If $A = c \oplus 1$ we eliminate A, B, C, the successors of B, x, and y (if C happens to be the only successor of B then it becomes constant and all its successors are eliminated). Hence $\{\Delta_0, \Delta_1\} = \{2 + \alpha, 4 + 2\alpha\}$. The inequalities eq. (10) are satisfied because of eq. (4).

Case 2.1.2. $\mathrm{out}(y) \geq 2$.

Denote by D another successor of y. Note that D might be equal to C, but $D \neq B, C$ by Transformation 4 of Subsection 3.4.2.



Case 2.1.2.1. B is trivialized when $A = c$.

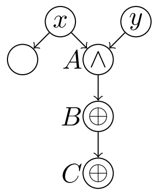
If $A = c$, we eliminate A, B, and the successors of B. If $A = c \oplus 1$, we eliminate A, B, C, D, x, and y. If $C = D$ then this gate becomes constant so all its successors are also eliminated. Hence $\{\Delta_0, \Delta_1\} = \{3, 4 + 2\alpha\}$. The inequalities eq. (10) are satisfied because eq. (7).

Case 2.1.2.2. B is trivialized when $A = c \oplus 1$.

If $A = c$, we eliminate A and B. If $A = c \oplus 1$, we eliminate A, B, C, D, the successors of B, x, and y. If $C \neq D$ and, say, C is a successor of B, then C becomes constant so all its successors are eliminated too. If $C = D$, then C becomes constant so all its successors are eliminated. Hence $\{\Delta_0, \Delta_1\} = \{2, 5 + 2\alpha\}$. The inequality eq. (5) ensures eq. (10).

Case 2.2. $\mathrm{out}(A) = 1$ and A feeds an xor-gate B.

Case 2.2.1. $\mathrm{out}(B) = 1$ and B feeds an xor-gate C.

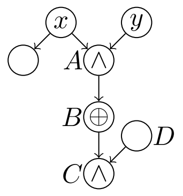


Since A is and-gate with the maximal number of and-gates on a way to the output of the circuit, we know that the gate C computes a function of degree at most two.. We make C constant. In both cases we eliminate A, B, C, and the successors of C. Hence $\Delta_0 = \Delta_1 = 4$. The inequalities eq. (10) are satisfied because of eq. (8).

Case 2.2.2. $\mathrm{out}(B) = 1$ and B feeds an and-gate C.

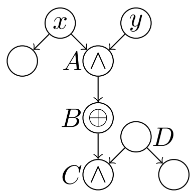
Let D be the other input of C. Note that if $D = A$ then the circuit is not optimal (C depends on A and the other input of B so one can compute C directly without using B).

Case 2.2.2.1. $\mathrm{out}(D) = 1$.



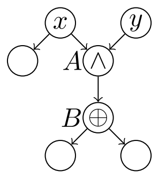
We make B constant. In both cases we eliminate A, B, and C. Moreover, when B is the constant trivializing C we eliminate also D and the successors of C. The gate D contributes (to the complexity decrease) $\alpha \le 1$ if it is an input gate and 1 if it is not an input. Hence we have $\{\Delta_0, \Delta_1\} = \{3, 4+\alpha\}$. The inequality eq. (7) guarantees that eq. (10) is satisfied.

Case 2.2.2.2. $\mathrm{out}(D) \ge 2$.



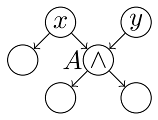
The gate D computes a function of degree at most 2: $out(A) = 1$, hence there is no path from A to D; if D computed a function of degree at most 3, A would not be a gate with the maximum number of and-type gates on a path to the output. We make D constant. In both cases we eliminate D and its successors and reduce the measure by at least $2+\alpha$ (as D might be an input). In the case when C becomes constant we eliminate also the successors of C as well as A and B. Thus, $\{\Delta_0, \Delta_1\} = \{2+\alpha, 5+\alpha\}$ (to ensure that all the five gates eliminated in the second case are different one notes that if D feeds B or a successor of C then the circuit is not optimal). The inequalities eq. (10) are satisfied because eq. (4) and $\alpha \le 1$.

Case 2.2.3. $\mathrm{out}(B) \ge 2$.



The gate B computes a polynomial of degree at most 2. By making it constant we eliminate B, its successors, and A, so $\Delta_0 = \Delta_1 = 4$. The inequalities eq. (10) are satisfied because of eq. (8).

Case 2.3. $\mathrm{out}(A) \ge 2$.



We make A constant. In both cases A and its successors are eliminated. When x and y become constant too (recall that if A computes $(x \oplus c_1)(y \oplus c_2) \oplus c$ then $A = c \oplus 1$ implies that $x = c_1 \oplus 1$ and $y = c_2 \oplus 1$) at least one other successor of x is also eliminated. Thus, $\{\Delta_0, \Delta_1\} = \{3, 4+2\alpha\}$. (If a successor of x coincides with a successor of A, it becomes constant and all its successors are also eliminated.) The inequality eq. (7) implies that eq. (10) is satisfied.

□

5 Conclusion and Open Problems

In this paper we broke the "$3n$ barrier" that stayed for thirty years after Blum's paper [6] and demonstrated that there is no combinatorial obstacle in the constant 3 here. Both our proofs use gate elimination and decompose the circuit into $D \circ L$, where L is an xor-circuit. It would be interesting to use a different class for the decomposition, for example, "quadratic" circuits having at most one nonlinear gate on each path.

Gate elimination alone has been recently shown to be unable to prove nonlinear bounds [19] because there are "robust" functions of high complexity that survive many (arbitrary) substitutions losing their complexity (measured as the number of gates or any other subadditive function) only proportional to the number of substituted variables. We must, however, notice that this result does not exclude additional tricks based on specific functions properties (for example, it is unclear whether there are "robust" affine dispersers), and not all the complexity measures are obviously subadditive. In addition, other types of breaking the function rather than substituting subsequent variables are available. For example, one can assign a specific value to an appropriate gate in the middle of a circuit as it is done in [49] (see also [38]) for proving a CircuitSAT upper bound. This can be viewed as going deep into a very complicated variety, which breaks completely the argument of [19].

Finally, finding explicit constructions of quadratic dispersers would transfer the bounds of Section 4 into the status of lower bounds for explicit functions. Constructions of dispersers for higher degrees will give more motivation for developing the techniques of this paper beyond "simple" varieties.

Acknowledgments

We would like to thank Dmitry Itsykson and Alexander Knop for their valuable comments, and Olga Melanich for proofreading an earlier version of this paper. We are thankful to Tianqi Yang and Jiatu Li for pointing out a gap (and proposing a solution) in the Case 7.2.4.2 in Subsection 3.5. We are very grateful to the anonymous reviewers whose suggestions and comments significantly helped to improve the presentation in this paper.

References

[1] Kazuyuki Amano and Jun Tarui. A well-mixed function with circuit complexity 5n: Tightness of the lachish-raz-type bounds. *Theor. Comput. Sci.*, 412(18):1646–1651, 2011. doi: 10.1016/j.tcs.2010.12.040. URL http://dx.doi.org/10.1016/j.tcs.2010.12.040.

[2] Alexander 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] Eli Ben-Sasson and Ariel Gabizon. Extractors for polynomials sources over constant-size fields of small characteristic. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 399–410. Springer, 2012.

[4] Eli Ben-Sasson and Swastik Kopparty. Affine dispersers from subspace polynomials. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, volume 679, pages 65–74. ACM Press, 2009. ISBN 9781605585062. doi: 10.1145/1536414.1536426. URL http://portal.acm.org/citation.cfm?doid=1536414.1536426.

[5] Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 163–173. Springer, 2014. ISBN 978-3-662-43947-0. doi: 10.1007/978-3-662-43948-7_14. URL http://dx.doi.org/10.1007/978-3-662-43948-7.

[6] Norbert Blum. A Boolean function requiring $3n$ network size. *Theor. Comput. Sci.*, 28:337–345, 1984. doi: 10.1016/0304-3975(83)90029-4. URL http://dx.doi.org/10.1016/0304-3975(83)90029-4.

[7] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *CCC-98*, 1998.

[8] Jin-Yi Cai. $S_2 \subseteq ZPP^{NP}$. In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 620–628, Oct 2001.

[9] Eshan Chattopadhyay, Jesse Goodman, and Jyun-Jie Liao. Affine extractors for almost logarithmic entropy. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 622–633. IEEE, 2022.

[10] Ruiwen Chen and Valentine Kabanets. Correlation bounds and #sat algorithms for small linear-size circuits. In Dachuan Xu, Donglei Du, and Dingzhu Du, editors, *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, volume 9198 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2015. ISBN 978-3-319-21397-2. doi: 10.1007/978-3-319-21398-9_17. URL http://dx.doi.org/10.1007/978-3-319-21398-9.

[11] Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015. doi: 10.1007/s00037-015-0100-0. URL http://dx.doi.org/10.1007/s00037-015-0100-0.

[12] Gil Cohen and Igor Shinkar. The complexity of DNF of parities. Technical Report 99, Electronic Colloquium on Computational Complexity, 2014.

[13] Gil Cohen and Avishay Tal. Two structural results for low degree polynomials and applications. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 680–709. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. ISBN 978-3-939897-89-7. doi: 10.4230/LIPIcs.APPROX-RANDOM.2015.680. URL http://dx.doi.org/10.4230/LIPIcs.APPROX-RANDOM.2015.680.

[14] Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In *Proceedings of 36th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6907 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 2011.

[15] Evgeny Demenkov, Alexander S. Kulikov, Olga Melanich, and Ivan Mihajlin. New lower bounds on circuit size of multi-output functions. *Theory Comput. Syst.*, 56(4):630–642, 2015. doi: 10.1007/s00224-014-9590-4. URL http://dx.doi.org/10.1007/s00224-014-9590-4.

[16] Zeev Dvir. Extractors for varieties. *Computational complexity*, 21(4):515–572, 2012.

[17] Patrick W. Dymond and Stephen A. Cook. Complexity theory of parallel time and hardware. *Inf. Comput.*, 80(3):205–226, 1989. doi: 10.1016/0890-5401(89)90009-6. URL http://dx.doi.org/10.1016/0890-5401(89)90009-6.

[18] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009. doi: 10.1145/1552285.1552286. URL http://doi.acm.org/10.1145/1552285.1552286.

[19] Alexander Golovnev, Edward A. Hirsch, Alexander Knop, and Alexander S. Kulikov. On the Limits of Gate Elimination. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:13, 2016.

[20] Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986. ISBN 0-89791-193-8. doi: 10.1145/12130.12132. URL http://doi.acm.org/10.1145/12130.12132.

[21] Johan Håstad. The shrinkage exponent of de Morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998. doi: 10.1137/S0097539794261556. URL http://dx.doi.org/10.1137/S0097539794261556.

[22] Erhard Heinz. Beiträge zur störungstheorie der spektralzerleung. *Mathematische Annalen*, 123(1): 415–438, 1951.

[23] Russell Impagliazzo and Noam Nisan. The effect of random restrictions on formula size. *Random Struct. Algorithms*, 4(2):121–134, 1993. doi: 10.1002/rsa.3240040202. URL http://dx.doi.org/10.1002/rsa.3240040202.

[24] Russell Impagliazzo, Valentine Kabanets, and Ilya Volkovich. The power of natural properties as oracles. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 102. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[25] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of $5n - o(n)$ for Boolean circuits. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2002. ISBN 3-540-44040-2. doi: 10.1007/3-540-45687-2_29.

[26] Tianqi Yang Jiatu Li. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *Proceedings on 54th Annual ACM Symposium on Theory of Computing*, 2022. to appear.

[27] Valeriy M. Khrapchenko. A method of determining lower bounds for the complexity of π-schemes. *Math. Notes of the Acad. of Sci. of the USSR*, 10(1):474–479, 1971.

[28] Boris M. Kloss and Vadim A. Malyshev. Estimates of the complexity of certain classes of functions. *Vestn.Moskov.Univ.Ser.1*, 4:44–51, 1965. In Russian.

[29] Donald E. Knuth. *The Art of Computer Programming*, volume 4, pre-fascicle 6a. Addison–Wesley, 2015. Section 7.2.2.2. Satisfiability. Draft available at http://www-cs-faculty.stanford.edu/~uno/fasc6a.ps.gz.

[30] Arist Kojevnikov and Alexander 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, Miami, Florida, USA, January 22-26, 2006*, pages 11–17. ACM Press, 2006. ISBN 0-89871-605-5. URL http://dl.acm.org/citation.cfm?id=1109557.1109559.

[31] Arist Kojevnikov and Alexander S. Kulikov. Circuit complexity and multiplicative complexity of Boolean functions. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes, 6th Conference on Computability in Europe, CiE 2010*, volume 6158 of *Lecture Notes in Computer Science*, pages 239–245. Springer, 2010. ISBN 978-3-642-13961-1. doi: 10.1007/978-3-642-13962-8_27. URL http://dx.doi.org/10.1007/978-3-642-13962-8.

[32] Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for demorgan formula size. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 588–597. IEEE Computer Society, 2013. ISBN 978-0-7695-5135-7. doi: 10.1109/FOCS.2013.69. URL http://dx.doi.org/10.1109/FOCS.2013.69.

[33] Alexander S. Kulikov and Nikita Slezkin. Sat-based circuit local improvement. *CoRR*, abs/2102.12579, 2021. URL https://arxiv.org/abs/2102.12579.

[34] Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theor. Comput. Sci.*, 223(1-2):1–72, 1999. doi: 10.1016/S0304-3975(98)00017-6. URL http://dx.doi.org/10.1016/S0304-3975(98)00017-6.

[35] Oded Lachish and Ran Raz. Explicit lower bound of $4.5n - o(n)$ for Boolean circuits. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 399–408. ACM, 2001. ISBN 1-58113-349-9. doi: 10.1145/380752.380832. URL http://doi.acm.org/10.1145/380752.380832.

[36] Xin Li. A new approach to affine extractors and dispersers. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, June 8-10, 2011*, pages 137–147. IEEE Computer Society, 2011. doi: 10.1109/CCC.2011.27. URL http://dx.doi.org/10.1109/CCC.2011.27.

[37] Xin Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 168–177. IEEE, 2016.

[38] A. A. Lialina. On the complexity of unique circuit SAT. *Zap. Nauchn. Sem. S.-Peterburg. Otdel. Mat. Inst. Steklov. (POMI)*, 475(Kombinatorika i Teoriya Grafov. X):122–136, 2018. ISSN 0373-2703.

[39] Edward I. Nechiporuk. On a Boolean function. *Doklady Akademii Nauk. SSSR*, 169(4):765–766, 1966.

[40] Arfst Nickelsen, Till Tantau, and Lorenz Weizsäcker. Aggregates with component size one characterize polynomial space. *Electronic Colloquium on Computational Complexity (ECCC)*, 028, 2004. URL http://eccc.hpi-web.de/eccc-reports/2004/TR04-028/index.html.

[41] Sergey Nurk. An $2^{0.4058m}$ upper bound for Circuit SAT. Technical Report 10, Steklov Institute of Mathematics at St.Petersburg, 2009. PDMI Preprint.

[42] Mike Paterson and Uri Zwick. Shrinkage of de Morgan formulae under restriction. *Random Struct. Algorithms*, 4(2):135–150, 1993. doi: 10.1002/rsa.3240040203. URL http://dx.doi.org/10.1002/rsa.3240040203.

[43] Wolfgang J. Paul. A $2.5n$-lower bound on the combinational complexity of Boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977. doi: 10.1137/0206030. URL http://dx.doi.org/10.1137/0206030.

[44] Alexander A. Razborov. Lower bound on monotone complexity of some Boolean functions. *Doklady Akademii Nauk. SSSR*, 281(4):798–801, 1985.

[45] Marc D. Riedel and Jehoshua Bruck. Cyclic boolean circuits. *Discrete Applied Mathematics*, 160 (13-14):1877–1900, 2012. doi: 10.1016/j.dam.2012.03.039. URL http://dx.doi.org/10.1016/j.dam.2012.03.039.

[46] Ronald L. Rivest. The necessity of feedback in minimal monotone combinational circuits. *IEEE Trans. Computers*, 26(6):606–607, 1977. doi: 10.1109/TC.1977.1674886. URL http://doi.ieeecomputersociety.org/10.1109/TC.1977.1674886.

[47] Rahul Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

[48] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 183–192. IEEE Computer Society, 2010. ISBN 978-0-7695-4244-7. doi: 10.1109/FOCS.2010.25. URL http://dx.doi.org/10.1109/FOCS.2010.25.

[49] Sergey Savinov. Upper bounds for the Boolean circuit satisfiability problem. Master Thesis defended at St.Petersburg Academic University of Russian Academy of Sciences, 2014. In Russian.

[50] Claus-Peter Schnorr. Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. *Computing*, 13(2):155–171, 1974. doi: 10.1007/BF02246615. URL http://dx.doi.org/10.1007/BF02246615.

[51] Claus-Peter Schnorr. The combinational complexity of equivalence. *Theor. Comput. Sci.*, 1(4):289–295, 1976. doi: 10.1016/0304-3975(76)90073-6. URL http://dx.doi.org/10.1016/0304-3975(76)90073-6.

[52] Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013. doi: 10.1007/s00037-013-0067-7. URL http://dx.doi.org/10.1007/s00037-013-0067-7.

[53] Ronen Shaltiel. Dispersers for affine sources with sub-polynomial entropy. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 247–256. IEEE Computer Society, 2011. ISBN 978-1-4577-1843-4. doi: 10.1109/FOCS.2011.37. URL http://dx.doi.org/10.1109/FOCS.2011.37.

[54] Ronen Shaltiel. An introduction to randomness extractors. In *Proceedings of 38th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6756 of *Lecture Notes in Computer Science*, pages 21–41. Springer, 2011.

[55] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.

[56] Victor Shoup and Roman Smolensky. Lower bounds for polynomial evaluation and interpolation problems. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 378–383. IEEE Computer Society, 1991. ISBN 0-8186-2445-0. doi: 10.1109/SFCS.1991.185394. URL http://dx.doi.org/10.1109/SFCS.1991.185394.

[57] Larry J. Stockmeyer. On the combinational complexity of certain symmetric Boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977. doi: 10.1007/BF01683282. URL http://dx.doi.org/10.1007/BF01683282.

[58] Bella A. Subbotovskaya. Realizations of linear functions by formulas using $+, \cdot, -$. *Doklady Akademii Nauk. SSSR*, 136(3):553–555, 1961.

[59] Avishay Tal. Shrinkage of de Morgan formulae by spectral techniques. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 551–560. IEEE, 2014.

[60] Salil Vadhan and Ryan Williams. Personal communication, 2013.

[61] E. Warning. Bemerkung zur vorstehenden Arbeit von Herrn Chevalley. *Abh. Math. Sem. Univ. Hamburg*, 11(1):76–83, 1935.

[62] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. Extended abstract appeared in Proc. STOC-2010.

[63] Ryan Williams. Nonuniform ACC circuit lower bounds. *JACM*, 61(1), 2014. Extended abstract appears in Proc. CCC-2011.

[64] Andrew C. Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *FOCS*, pages 1–10. IEEE Computer Society, 1985. ISBN 0-8186-0644-4. URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4568115.

[65] Amir Yehudayoff. Affine extractors over prime fields. *Combinatorica*, 31(2):245–256, 2011. doi: 10.1007/s00493-011-2604-9. URL http://dx.doi.org/10.1007/s00493-011-2604-9.

[66] Uri 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.*, 20(3):499–505, 1991. doi: 10.1137/0220032. URL http://dx.doi.org/10.1137/0220032.