Circuit Complexity: New Techniques and Their Limitations

by

Aleksandr Golovnev

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NEW YORK UNIVERSITY

MAY, 2017

Yevgeniy Dodis

Oded Regev

Dedication

to Ludmila Golovneva

Acknowledgments

I would like to express my sincere gratitude to my advisors, Yevgeniy Dodis and Oded Regev, who introduced me to the beautiful world of theoretical computer science. I thank them for their immense knowledge, continuous support, encouragement and motivation. Most of what I know I learned from long discussions with Oded and from his great and insightful advice. I thank my committee, Subhash Khot, Rocco Servedio, and Ryan Williams for their kind help and comments which greatly improved this work.

I am grateful to all my co-authors, with whom working was extremely pleasant and informative: Magnus Gausdal Find, Edward A. Hirsch, Alexander Knop, Alexander S. Kulikov, Alexander Smal, and Suguru Tamaki.

I thank Alexander S. Kulikov for his endless encouragement, optimism, and very patient mentoring over many years. Without him no part of this work would have been possible. I am extremely grateful to my teacher of mathematics, Boris Komrakov, for his illuminating lectures years ago, and for his still constant support. I owe many thanks to Ryan Williams whose brilliant work on circuits and algorithms inspired and determined my research interests.

I would like to thank the computer science department for their help, especially Rosemary Amico and Santiago Pizzini, who were always very friendly and

helped me on many occasions.

I owe many thanks to my friends from Courant for creating an encouraging, enjoyable (and sometimes) working environment. Our student seminars, discussions and more importantly bets were always fun. I thank them for their continuous support, cheer and pi-jaws: Azam Asl, Huck Bennett, Sandro Coretti, Laura Florescu, Chaya Ganesh, Sid Krishna, Shravas Rao, Noah Stephens-Davidowitz, and Deva Thiruvenkatachari.

Finally, I would like to thank my wife Olya, my mother Alla, and my brother Arseniy for inspiring me to study mathematics, and for their continuous love and encouragement.

Abstract

We study the problem of proving circuit lower bounds. 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 a function from **NP** requiring even a superlinear number of gates. Moreover, only modest linear lower bounds are known. Until this work, the strongest known lower bound was $3n - o(n)$ (Blum 1984).

Essentially, the only known method for proving lower bounds on general Boolean circuits is gate elimination. We extend and generalize this method in order to get stronger circuit lower bounds, and we get better algorithms for the Circuit SAT problem. We also study the limitations of gate elimination.

- We extend gate elimination to prove a lower bound of $\left(3 + \frac{1}{86}\right) n - o(n)$ for the circuit size of an affine disperser for sublinear dimension. There are known explicit constructions of such functions.

- We introduce the weighted gate elimination method, which runs a more sophisticated induction than gate elimination. This method gives a much simpler proof of a stronger lower bound of $3.11n$ for quadratic dispersers. Currently, there is no known example of a quadratic disperser in **NP** (al-

though there are constructions that work for parameters different than the ones that we need).

- The most technical part of gate elimination proofs is the case analysis. We develop a general framework which allows us to reuse the same case analysis for proving worst-case and average-case circuit lower bounds, and upper bounds for Circuit SAT algorithms. Using this framework we improve known upper bounds for Circuit SAT, and prove a stronger average-case lower bound for the circuit basis U_2.

- We study the limits of gate elimination proofs. We show that there exists an explicit constant c, such that the current techniques used in gate elimination cannot prove a linear lower bound of cn.

Contents

List of Figures

1

Introduction

1.1 Overview

Let us consider a Boolean function of n arguments $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$. A natural question studied in theoretical computer science is: 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 source vertices (called inputs) $x_1, \ldots, x_n$, whose intermediate vertices (called gates) have indegree 2 and are labeled with arbitrary binary Boolean operations. The size of a circuit is its number of gates. Note that we do not impose any restrictions on the depth or outdegree.

Counting shows that the number of circuits with small size is much smaller than the total number 2^{2^n} of Boolean functions of n arguments. Using this idea it was shown by Shannon [100] that almost all functions of n arguments 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 functions from **NP**) remains a difficult problem. (In particular, such lower bounds would imply $\mathbf{P} \neq \mathbf{NP}$.) Moreover, even superlinear bounds are unknown for functions in $\mathbf{E}^{\mathbf{NP}}$. Superpolynomial bounds are known for **MAEXP** (exponential-time Merlin-Arthur games) [16] and $\mathbf{ZPEXP}^{\mathbf{MCSP}}$ (exponential-time **ZPP** with oracle access to the Minimal Circuit Size Problem) [49], and arbitrary polynomial lower bounds are known for $\mathbf{O_2}$ (the oblivious symmetric second level of the polynomial hierarchy) [17].

People started to tackle this problem in the 60s. Kloss and Malyshev [57] proved a $2n - O(1)$ lower bound for the function $\bigoplus_{1\leq i<j\leq n} x_i x_j$. Schnorr [95] proved a $2n - O(1)$ lower bound for a class of functions with certain structure. Stockmeyer [102] proved a $2.5n - O(1)$ bound for most symmetric functions. Paul [83] proved a $2n - o(n)$ lower bound for the storage access function and a $2.5n - o(n)$ lower bound for a combination of two storage access functions. Eventually, in 1984 Blum [14] extended Paul's argument and proved a $3n - o(n)$ bound for a function combining three storage access functions using simple operations.

Blum's bound remained unbeaten for more than thirty years. Blum's proof relies on a number of properties of his particular function, and it cannot be ex-

tended to get a stronger than $3n$ lower bound without using different properties.

Recently, Demenkov & Kulikov [29] presented a much simpler proof of a $3n - o(n)$ lower bound for functions with an entirely different property: affine dispersers (and there are known efficient constructions of affine dispersers in **P**). This property allows one to restrict the function to smaller and smaller affine subspaces. As was later observed by Vadhan & Williams [106], the way Demenkov and Kulikov use this property cannot give stronger than $3n$ bounds as it is tight for the inner product function.* (But this does not extinguish all hope of using affine dispersers to prove better lower bounds.) Hence, mysteriously, two different proofs using two different properties are both stuck on exactly the same lower bound $3n - o(n)$ which was first proven more than 30 years ago. Is this lack of progress grounded in combinatorial properties of circuits, so that this line of research faces an insurmountable obstacle? Or can refinements of the known techniques go above $3n$?

In this work 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 proven 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 over other variables of degree at most 2. Currently, there are no examples of quadratic dispersers in **NP** (though there are constructions with weaker parameters for the field of size two and constructions for larger fields).

*The inner product function is known to be an affine disperser for dimension $n/2 + 1$.

We also study applications of these techniques to algorithms for the Circuit Satisfiability problem, and give evidence that these techniques cannot lead to strong linear lower bounds.

1.2 Computational models

The exact complexity of computational problems is different in different models of computation. For example, switching from multitape to single-tape Turing machines can square the time complexity, and 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. Furthermore, finding a function hard for Boolean circuits can be viewed as a combinatorial problem, in contrast to lower bounds for uniform models (models with 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.

In this work we consider single-output circuits (that is, circuits computing Boolean predicates). It would be natural to expect functions with larger output to lead to stronger bounds. However, the only tool we have to transfer bounds from one output to several outputs is Lamagna's and Savage's[66] argument showing that in order to compute simultaneously m different functions requiring c gates each, one needs at least $m + c - 1$ gates. That is, we do not

have superlinear bounds for multioutput functions either.

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). With this restricted basis, Schnorr[96] proved that the circuit complexity of the parity function is $3n - 3$. Zwick[118] gave a $4n - O(1)$ lower bound for certain symmetric functions, Lachish & Raz[65] 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 & Morizumi[53] improved this bound to $5n - o(n)$. Demenkov et al.[31] gave a simpler proof of a $5n - o(n)$ lower bound for a function with $o(n)$ outputs. It is interesting to note that the progress on U_2 circuit lower bounds is also stuck on the $5n - o(n)$ lower bound: Amano & Tarui[6] 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, exponential bounds are known for weaker models: one thread was initiated by Razborov[85] for monotone circuits; another one was started by Yao and Håstad[116,44] for constant-depth circuits with unbounded fanin AND/OR gates and NOT gates. Shoup and Smolensky[101] proved a superlinear lower bound $\Omega(n \log n / \log \log n)$ for linear circuits of polylogarithmic depth over infinite fields. Also, superlinear bounds for formulas have been known for half a century. For de Morgan formulas (i.e., formulas over AND, OR, NOT) Subbotovskaya[103] proved an $\Omega(n^{1.5})$ lower bound for the parity function using the random restrictions method. Khrapchenko[56] showed an $\Omega(n^2)$ lower bound for

parity. Applying Subbotovskaya's random restrictions method to the universal function by Nechiporuk[74], Andreev[7] proved an $\Omega(n^{2.5-o(1)})$ lower bound. By analyzing how de Morgan formulas shrink under random restrictions, Andreev's lower bound was improved to $\Omega(n^{2.55-o(1)})$ by Impagliazzo and Nisan[51], then to $\Omega(n^{2.63-o(1)})$ by Paterson and Zwick[81], and eventually to $\Omega(n^{3-o(1)})$ by Håstad[45] and Tal[104]. For formulas over the full binary basis, Nechiporuk[74] proved an $\Omega(n^{2-o(1)})$ lower bound for the universal function and for the element distinctness function. These bounds, however, do not translate to superlinear lower bounds for general constant-arity Boolean circuits.

1.3 Circuit SAT algorithms

A recent promising direction initiated by Williams[111,115] suggests the following approach for proving circuit lower bounds against E^{NP} or NE using SAT-algorithms: a super-polynomially faster-than-2^n time algorithm for the circuit satisfiability problem of a "reasonable" circuit class $\mathcal{C}$ implies either $\mathrm{E}^{\mathrm{NP}} \not\subseteq \mathcal{C}$ or $\mathrm{NE} \not\subseteq \mathcal{C}$, depending on $\mathcal{C}$ and the running time of the algorithm. In this way, unconditional exponential lower bounds have been proven for ACC_0 circuits (constant-depth circuits with unbounded-arity OR, AND, NOT, and arbitrary cosntant modular gates)[115]. The approach has been strengthened and simplified by subsequent work[109,112,114,13,54], see also excellent surveys[93,80,113] on this topic.

Williams' result inspired lots of work on satisfiability algorithms for various circuit classes[52,114,22,5,4,73,23,105]. In addition to satisfiability algorithms, several papers[92,50,10,97,21,19,21,24,91] also obtained average-case lower bounds (also known

as correlation bounds, see[61,62,46]) by investigating the analysis of algorithms instead of just applying Williams' result for worst-case lower bounds.

It should be noted, however, that currently available algorithms for the satisfiability problem for general circuit classes are not sufficient for proving many lower bounds. Current techniques require algorithmic upper bounds of the form $O(2^n/n^a)$ for circuits with n inputs and size n^k, while for most circuit classes only c^g-time algorithms are available, where g is the number of the gates and $c > 1$ is a constant.

On the other hand, the techniques used in the c^g-time algorithms for CircuitSAT are somewhat similar to the techniques used for proving linear lower bounds for (general) Boolean circuits over the full binary basis. In particular, an $O(2^{0.4058g})$-time algorithm by Nurk[79] (and subsequently an $O(2^{0.389667g})$-time algorithm by Savinov[94]) used a reconstruction of the linear part of a circuit similar to the one suggested by Paul[83]. These algorithms and proofs use similar tricks in order to simplify circuits.

Chen and Kabanets[20] presented algorithms that count the number of satisfying assignments of circuits over U_2 and B_2 and run in time exponentially faster than 2^n if input instances have at most $2.99n$ and $2.49n$ gates, respectively (improving also the previously best known #SAT-algorithm by Nurk[79]). At the same time, they showed that $2.99n$ sized circuits over U_2 and $2.49n$ sized circuits over B_2 have exponentially small correlations with the parity function and affine extractors with "good" parameters, respectively.

Generalizing this work, we also provide a general framework which takes

a gate-elimination proof and constructs a proof of worst/average case lower bounds for circuits and upper bounds for #SAT.

1.4 Known limitations for proving lower bounds

Although there is no known argument limiting the power of gate elimination, there are many known barriers to proving circuit lower bounds. In this section we list some of them. This list does not pretend to cover all known barriers in proving lower bounds, but we try to show both fundamental barriers in proving strong bounds and limits of specific techniques.

1.4.1 Circuit lower bounds

Baker, Gill, and Solovay[9,39] present the *relativization* barrier that shows that any solution to the P versus NP question must be non-relativizing. In particular, they show that the classical diagonalization technique is not powerful enough to resolve this question. Aaronson and Wigderson[1] present the *algebrization* barrier that generalizes relativization. For instance, they show that any proof of superlinear circuit lower bound requires non-algebrizing techniques. The *natural proofs* argument by Razborov and Rudich[88] shows that a "natural" proof of a circuit lower bound would contradict the conjecture that strong one-way functions exist. This rules out many approaches; for example, this argument shows that the *random restrictions* method[44] is unlikely to prove superpolynomial lower bounds. The natural proofs argument implies the following limitation for the gate elimination method. If subexponentially strong one-way

functions exist, then for any large class $\mathcal{P}$ of functions (i.e., a class with at least a $\frac{1}{n}$ fraction of the languages in $\mathcal{P}$), for any effective measure (computable in time $2^{O(n)}$) and effective family of substitutions $\mathcal{S}$ (i.e., a family of substitutions enumerable in time $2^{O(n)}$), gate elimination using the family $\mathcal{S}$ of substitutions cannot prove lower bounds better than $O(n)$. We note that the measures considered in this work are not known to be effective.

Let $\mathcal{F}$ be a family of Boolean functions of n variables. Let X and Y be disjoint sets of input variables, and $|X| = n$. Then a Boolean function $UF(X, Y)$ is called *universal* for the family $\mathcal{F}$ if for every $f(X) \in \mathcal{F}$, there exists an assignment c of constants to the variables Y, such that $UF(X, c) = f(X)$. For example, it can be shown that the function used by Blum[14] is universal for the family $\mathcal{F} = \{x_i \oplus x_j, x_i \wedge x_j | 1 \leq i, j \leq n\}$. Nigmatullin[77,78] shows that many known proofs can be stated as lower bounds for universal functions for families of low-complexity functions. At the same time, Valiant[107] proves a linear upper bound on the circuit complexity of universal functions for these simple families.

There are known linear upper bounds on circuit complexity of some specific functions and even classes of functions. For example, Demenkov et al.[28] show that each *symmetric function* (i.e., a function that depends only on the sum of its inputs over the integers) can be computed by a circuit of size $4.5n + o(n)$. This, in turn, implies that no gate elimination argument for a class of functions that contains a symmetric function can lead to a superlinear lower bound.

The basis U_2 is the basis of all binary Boolean functions without parity and its negation. The strongest known lower bound for circuits over the basis U_2 is

$5n - o(n)$. This bound is proved by Iwama and Morizumi[53] for $(n - o(n))$-*mixed* functions. Amano and Tarui[6] construct an $(n - o(n))$-mixed function whose circuit complexity over U_2 is $5n + o(n)$.

1.4.2 Formula lower bounds

A formula is a circuit where each gate has out-degree one. The best known lower bound of $n^{2-o(1)}$ on formula size was proven by Nechiporuk[74]. The proof of Nechiporuk is based on counting different *subfunctions* of the given function. It is known that this argument cannot lead to a superquadratic lower bound (see, e.g., Section 6.5 in[55]).

A De Morgan formula is a formula with AND and OR gates, whose inputs are variables and their negations. The best known lower bound for De Morgan formulas is $n^{3-o(1)}$ (Håstad[45], Tal[104], Dinur and Meir[32]). The original proof of this lower bound by Håstad is based on showing that the shrinkage exponent Γ is at least 2. This cannot be improved since Γ is also at most 2 as can be shown by analyzing the formula size of the parity function.

Paterson introduces the notion of formal complexity measures for proving De Morgan formula size lower bounds (see, e.g.,[110]). A formal complexity measure is a function $\mu\colon B_n \to \mathbb{R}$ that maps Boolean functions to reals, such that

1. for every literal x, $\mu(x) \leq 1$;
2. for all Boolean functions f and g, $\mu(f \wedge g) \leq \mu(f) + \mu(g)$ and $\mu(f \vee g) \leq \mu(f) + \mu(g)$.

It is known that De Morgan formula size is the largest formal complexity measure. Thus, in order to prove a lower bound on the size of De Morgan formula, it suffices to define a formal complexity measure and show that an explicit function has high value of measure. Khrapchenko[56] uses this approach to prove an $\Omega(n^2)$ lower bound on the size of De Morgan formulas for parity. Unfortunately, many natural classes of formal complexity measures cannot lead to stronger lower bounds. Hrubes et al.[48] prove that *convex* measures (including the measure used by Khrapchenko) cannot lead to superquadratic bounds. A formula complexity measure μ is called *submodular*, if for all functions f, g it satisfies $\mu(f \vee g) + \mu(f \wedge g) \leq \mu(f) + \mu(g)$. Razborov[86] uses a submodular measure based on matrix parameters to prove superpolynomial lower bounds on the size of monotone formulas. In a subsequent work, Razborov[87] shows that submodular measures cannot yield superlinear lower bounds for non-monotone formulas. The *drag-along principle*[88,69] shows that no useful formal complexity measure can capture specific properties of a function. Namely, it shows that if a function has measure m, then a random function with probability $1/4$ has measure at least $m/4$. Measures based on graph entropy (Newman and Wigderson[75]) are used to prove a lower bound of $n \log n$ on De Morgan formula size, but it is proved that these measures cannot lead to stronger bounds.

1.4.3 Gate elimination

We study limits of the gate elimination proofs. A typical gate elimination argument shows that it is possible to eliminate several gates from a circuit by

making one or several substitutions to the input variables and repeats this inductively. In this work we prove that this method cannot achieve linear bounds of cn beyond a certain constant c, where c depends only on the number of substitutions made at a single step of the induction. We note that almost all known proofs make only one or two substitutions at a step. Thus, this limitation result has an explicit small constant c for them.

1.5 Outline

Chapter 2 provides notation and definitions used in this work, Chapter 3 defines the gate elimination method and gives an overview of our lower bounds proofs. In Chapter 4 we give a proof of a $(3 + \frac{1}{86})n - o(n)$ circuit lower bound. Chapter 5 introduces the weighted gate elimination method and presents a proof of a conditional lower bound of $3.11n$. Chapter 6 studies applications of the gate elimination method to average-case lower bounds and upper bounds for #SAT. Finally, Chapter 7 discusses limitations of the developed techniques.

Most of the results in this work appeared in the papers[37,41,42,40], and are based on joint works with Magnus Gausdal Find, Edward A. Hirsch, Alexander Knop, Alexander S. Kulikov, Alexander Smal, and Suguru Tamaki.

2

Preliminaries

2.1 Circuits

Let us denote by $B_{n,m}$ the set of all Boolean functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$, and let $B_n = B_{n,1}$. A circuit is an acyclic directed graph. A vertex in this graph may either have indegree zero (in which case it is called an *input* or a *variable*) or indegree two (in which case it is called a *gate*). Every gate is labelled by a Boolean function $g\colon \{0,1\} \times \{0,1\} \to \{0,1\}$, and the set of all the sixteen such functions is denoted by B_2.

For a circuit C, $G(C)$ is the number of gates and is also called the size of the circuit C. By $I(C)$ we denote the number of inputs, and by $I_1(C)$ the number

of inputs of out-degree 1. For a function $f \in B_{n,m}$, $C(f)$ is the minimum size of a circuit with n inputs and m outputs computing f.

We also consider the basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$ containing all binary Boolean functions except for the parity and its complement. For a function $f \in B_n$ and a basis Ω, by $C_\Omega(f)$ we denote the minimal size of a circuit over Ω computing f.

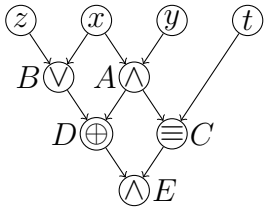
We say that a gate with inputs x and y is of *and-type* if it computes $g(x, y) = (c_1 \oplus x)(c_2 \oplus y) \oplus c_3$ for some constants $c_1, c_2, c_3 \in \{0, 1\}$, and of *xor-type* if it computes $g(x, y) = x \oplus y \oplus c_1$ for some constant $c_1 \in \{0, 1\}$. If a gate computes an operation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

If a gate computes a constant operation, we call it *trivial*. 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 (thus, we have 0-variables, 1-variables, 2^+-variables, etc.).

A toy example of a circuit is shown in Figure 2.1. For inputs, the corresponding variables are shown inside. For a 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.

For two Boolean functions $f, g \in B_n$, the correlation between them is defined



$$B = (z \vee x)$$
$$A = (x \wedge y)$$
$$D = (B \oplus A)$$
$$C = (A \equiv t)$$
$$E = (D \wedge C)$$

Figure 2.1: An example of a circuit and the program it computes.

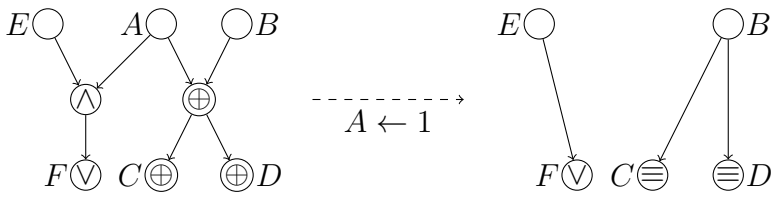
as

$$\begin{aligned} \mathrm{Cor}(f,g) &= \left| \Pr_{x \leftarrow \{0,1\}^n}[f(x) = g(x)] - \Pr_{x \leftarrow \{0,1\}^n}[f(x) \neq g(x)] \right| \\ &= 2\left| \frac{1}{2} - \Pr_{x \leftarrow \{0,1\}^n}[f(x) \neq g(x)] \right|. \end{aligned}$$

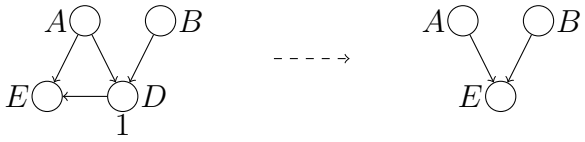
For a function $f \in B_n$, basis Ω and $0 \leq \epsilon \leq 1$, by $C_\Omega(f, \epsilon)$ we denote the minimal size of a circuit over Ω computing function g such that $\mathrm{Cor}(f, g) \geq \epsilon$.

2.1.1 Circuit normalization

When a gate (or an input) A of a circuit trivializes (e.g., when an input is assigned a constant), some other gates (in particular, all successors of A) may become trivial or degenerate. Such gates can be eliminated from the circuit without changing the function computed by the circuit (see an example below). Note that this simplification may change outdegrees and binary operations computed by other gates.



A gate is called *useless* if it is a 1-gate and is fed by a predecessor of its successor:



In this example the gate D is useless, and the gate E computes a binary operation of A and B, which can be computed without the gate D. This might require to change an operation at E (if this circuit is over U_2 then E still computes an and-type operation of A and B since an xor-type binary function requires three gates in U_2).

By *normalizing* a circuit we mean removing all gates that compute trivial or degenerate operations and removing all useless gates.

In the proofs we implicitly assume that if two gates are fed by the same variable then either there is no wire between them or each of the gates feeds also some other gate (otherwise, one of the gates would be useless).

2.2 Dispersers and extractors

Extractors are functions that take input from some specific distribution and output a bit that is distributed statistically close to uniform*. Dispersers are

*In this work, we consider only dispersers and extractors with one bit outputs.

a relaxation of extractors; they are only required to output a non-constant bit on "large enough" structured subsets of inputs. 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[99] for an excellent survey). The sources we are focused on in this work are affine sources and their generalization — sources for polynomial varieties. Affine dispersers have drawn much interest lately. In particular, explicit constructions of affine dispersers for dimension $d = o(n)$ have been constructed[12,117,67,98,11,68]. Dispersers for polynomial varieties over large fields were studied by Dvir[34], and dispersers over $\mathbb{F}_2$ were studied by Cohen & Tal[27].

Let $x_1, \ldots, x_n$ be Boolean variables, and $f \in B_{n-1}$ be a function of $n-1$ variables. We say that $x_i \leftarrow f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ is a substitution to the variable x_i.

Let $g \in B_n$ be a function, then the *restriction* of g under the substitution f is a function $h = (g|x_i \leftarrow f)$ of $n-1$ variables, such that $h(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = g(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$, where $x_i = f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Similarly, if $K \subseteq \{0,1\}^n$ is a subset of the Boolean cube, then the *restriction* of K under this substitution is $K' = (K|x_i \leftarrow f)$,

such that $(x_1, \ldots, x_n) \in K'$ if and only if $(x_1, \ldots, x_n) \in K$ and $x_i = f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

For a family of functions $\mathcal{F} = \{f : \{0,1\}^* \to \{0,1\}\}$ we define a set of corresponding substitutions $\mathcal{S}(\mathcal{F})$ that contains the following substitutions: for every $1 \leq i \leq n, c \in \{0,1\}, f \in \mathcal{F}$, $\mathcal{S}$ contains the substitution $x_i \leftarrow f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \oplus c$.

Let $\mathcal{S}$ be a set of substitutions. We say that a set $K \subseteq \{0,1\}^n$ is an $(\mathcal{S}, n, r)$-*source* if it can be obtained from $\{0,1\}^n$ by applying at most r substitutions from $\mathcal{S}$.

Definition 1. A function $f \in B_n$ is called an $(\mathcal{S}, n, r)$-*disperser* if it is not constant on every $(\mathcal{S}, n, r)$-source. A function $f \in B_n$ is called an $(\mathcal{S}, n, r, \epsilon)$-*extractor* if $|\Pr_{x \leftarrow K}[f(x) = 1] - 1/2| \leq \epsilon$ for every $(\mathcal{S}, n, r)$-source K.

Definition 1 is parameterized by a class of substitutions. In this work, we will often use dispersers for the set of affine or quadratic substitutions, and we give specific definitions of the corresponding dispersers below.

Definition 2 (affine disperser)**.** An *affine disperser* for dimension $d(n)$ is a family of functions $f_n \colon \mathbb{F}_2^n \to \mathbb{F}_2$ such that for all sufficiently large n, f_n is non-constant on any affine subspace of dimension at least $d(n)$.

There are known explicit constructions of affine dispersers:

Theorem 1 ([12,117,67,98,11,68])**.** There exist affine dispersers for dimension $d = o(n)$ in $\mathbf{P}$.

Definition 3 (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 $t \leq k$ polynomials of degree at most 2:

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

where p_i is a polynomial of degree at most 2, for each $1 \leq i \leq t$. Here k and s can be functions of n.

Definition 4 (quadratic disperser)**.** An (n,k,s)-*quadratic disperser* is a family of functions $f_n \colon \mathbb{F}_2^n \to \mathbb{F}_2$ such that for all sufficiently large n, f_n is non-constant on any (n,k)-quadratic variety $S \subseteq \mathbb{F}_2^n$ of size at least s.

Although we do not know explicit constructions of quadratic dispersers, we note that almost all functions from B_n are $(n, 2^{o(n)}, 2^{o(n)})$-quadratic dispersers.

Lemma 1. Let $\omega(1) \leq s \leq 2^{o(n)}, k = o\left(\frac{s}{n^2}\right)$. Let $D_n \in B_n$ be the set of (n,k,s)-quadratic dispersers. Then $\frac{|D_n|}{|B_n|} \to 1$ when $n \to \infty$.

Proof. There are $q = \frac{n(n+1)}{2} + 1 = \Theta(n^2)$ 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 $2^{qk} \cdot 2 \cdot 2^{2^n - s} = 2^{2^n} 2^{qk+1-s} = 2^{2^n} 2^{-\Theta(s)} = o\left(|B_n|\right)$. □

Cohen & Tal[27] prove that any affine disperser (extractor) is also a disperser (extractor) for polynomial varieties with slightly weaker parameters. In particular, their result, combined with the affine disperser by Li[68], gives an explicit construction of an $\left(n, \Theta\left(\frac{n}{\text{poly}(\log n)}\right), 2^{o(n)}\right)$-quadratic disperser. Two explicit constructions of extractors for varieties over large fields are given by Dvir[34]. For a similar, although different, notion of polynomial sources, explicit constructions of dispersers (extractors) are given by Dvir et al.[35] for large fields, and by Ben-Sasson & Gabizon[11] for constant-size fields.

2.3 Circuit complexity measures

A function μ mapping circuits to non-negative real values is called a *circuit complexity measure* if for any circuit C,

- normalization of C does not increase its measure[†], and
- if $\mu(C) = 0$ then C has no gates.

For a fixed circuit complexity measure μ, and function $f \in B_n$, we define $\mu(f)$ to be the minimum value of $\mu(C)$ over circuits C computing f. Similarly, we define $\mu(f, \epsilon)$ to be the minimum value of $\mu(C)$ over circuits C computing g such that $\text{Cor}(f, g) \geq \epsilon$.

[†]See Section 2.1.1 for the definition of circuit normalization.

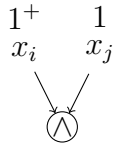
Such measures were previously considered by several authors. For example, Zwick[118] counted the number of gates minus the number of inputs of outdegree 1. The same measure was later used by Lachish & Raz[65] and by Iwama & Morizumi[53]. Kojevnikov & Kulikov[60] used a measure assigning different weights to linear and non-linear gates to show that Schnorr's $2n - O(1)$ lower bound[96] 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[63,59,38].

The following two circuit complexity measures will prove useful in Section 6:

- $\mu(C) = s(C) + \alpha \cdot i(C)$ where $\alpha \geq 0$ is a constant;
- $\mu(C) = s(C) + \alpha \cdot i(C) - \sigma \cdot i_1(C)$ where $0 \leq \sigma \leq 1/2 < \alpha$ are constants.

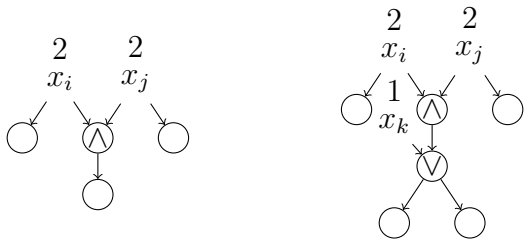
It is not difficult to see that these two functions are indeed circuit complexity measures. The condition $0 \leq \sigma \leq 1/2 < \alpha$ is needed to guarantee that if by removing a degenerate gate we increase the out-degree of a variable, the measure does not increase (an example is given below), and that the measure is always non-negative.

Intuitively, we include the term $I(C)$ into the measure to handle cases like the one below (throughout this work, we use labels above the gates to indicate their outdegrees, and we write k^+ to indicate that the degree is at least k):



In this case, by assigning $x_i \leftarrow 0$ we make the circuit independent of x_j, so the measure is reduced by at least 2α. Usually, our goal is to show that we can find a substitution to a variable that eliminates at least some constant number k of gates, that is, to show a complexity decrease of at least $k + \alpha$. Therefore, by choosing a large enough value of α we can always guarantee that $2\alpha \geq \alpha + k$. Thus, in the case above we do not even need to count the number of gates eliminated under the substitution.

The measure $\mu(C) = s(C) + \alpha \cdot I(C) - \sigma \cdot i_1(C)$ allows us to get an advantage of new 1-variables that are introduced during splitting.



For example, by assigning $x_i \leftarrow 0$ in a situation like the one in the left picture we reduce the measure by at least $3 + \alpha + \sigma$. As usual, the advantage comes with a related disadvantage. If, for example, a closer look at the circuit from the left part reveals that it actually looks as the one on the right, then by assigning $x_i \leftarrow 0$ we introduce a new 1-variable x_j, but also lose one 1-variable (namely, x_k is now a 2-variable). Hence, in this case μ is reduced only by $(3+\alpha)$ rather than $(3+\alpha+\sigma)$. That is, our initial estimate was too optimistic. For this reason, when use the measure with $I_1(C)$ we must carefully estimate the number of introduced 1-variables.

2.4 Splitting numbers and splitting vectors

Let μ be a circuit complexity measure and C be a circuit. Consider a recursive algorithm solving #SAT on C by repeatedly substituting input variables. Assume that at the current step the algorithm chooses k variables $x_1, \ldots, x_k$ and k functions $f_1, \ldots, f_k$ to substitute these variables and branches into 2^k situations: $x_1 \leftarrow f_1 \oplus c_1, \ldots, x_k \leftarrow f_k \oplus c_k$ for all possible $c_1, \ldots, c_k \in \{0,1\}$ (in other words, it partitions the Boolean hypercube $\{0,1\}^n$ into 2^k subsets).[‡] For each substitution, we normalize the resulting circuit. Let us call the 2^k nnormalized resulting circuits $C_1, \ldots, C_{2^k}$. We say that the current step has a *splitting vector* $v = (a_1, \ldots, a_{2^k})$ w.r.t. the circuit measure μ, if for all $i \in [2^k]$, $\mu(C) - \mu(C_i) \geq a_i > 0$. That is, the splitting vector gives a lower bound on the complexity decrease under the considered substitution. The *splitting number* $\tau(v)$ is the unique positive root of the equation $\sum_{i \in [2^k]} x^{-a_i} = 1$.

Splitting vectors and numbers are heavily used to estimate the running time of recursive algorithms. Below we assume that k is bounded by a constant. In all the proofs of this work either $k = 1$ or $k = 2$, that is, we always estimate the effect of assigning either one or two variables. If an algorithm always splits with a splitting number at most β then its running time is bounded by $O^*(\beta^{\mu(C)})$.[§] To show this, one notes that the recursion tree of this algorithm is 2^k-ary and $k = O(1)$ so it suffices to estimate the number of leaves. The number of leaves

[‡]Sometimes it is easier to consider vectors of length that is not a power of 2 too. For example, we can have a branching into three cases: one with one substituted variable, and two with two substituted variables. All the results from this work can be naturally generalized to this case. For simplicity, we state the results for splitting vectors of length 2^k only.

[§]O^* suppresses factors polynomial in the input length n.

$T(\mu)$ satisfies the recurrence $T(\mu) \leq \sum_{i\in[2^k]} T(\mu - a_i)$ which implies that $T(\mu) = O(\tau(v)^\mu)$ (we assume also that $T(\mu) = O(1)$ when $\mu = O(1)$). See, e.g.,[64] for a formal proof.

For a splitting vector $v = (a_1, \ldots, a_{2^k})$ we define the following related quantities:

$$\overline{v}_{\max} = \max_{i\in[2^k]} \left\{ \frac{a_i}{k} \right\}, \quad \overline{v}_{\min} = \min_{i\in[2^k]} \left\{ \frac{a_i}{k} \right\}, \quad \overline{v}_{\text{avg}} = \frac{\sum_{i\in[2^k]} a_i}{k2^k}.$$

Intuitively, $\overline{v}_{\max}$ ($\overline{v}_{\min}$, $\overline{v}_{\text{avg}}$) is a (lower bound for) the maximum (minimum, average, respectively) complexity decrease per single substitution.

We will need the following estimates for the splitting numbers. It is known that a balanced binary splitting vector is better than an unbalanced one: $2^{1/a} = \tau(a, a) < \tau(a + b, a - b)$ for $0 < b < a$ (see, e.g.,[64]). There is a known upper bound on $\tau(a, b)$.

Lemma 2. $\tau(a, b) \leq 2^{1/\sqrt{ab}}$.

In the following lemma we provide an asymptotic estimate of their difference.

Lemma 3 (Gap between $\tau(a_1 + b, a_2 + b)$ and $\tau((a_1 + a_2)/2 + b, (a_1 + a_2)/2 + b)$)**.** Let $a_1 > a_2 > 0$, $a' = (a_1 + a_2)/2$ and $\delta(b) = \tau(a_1 + b, a_2 + b) - 2^{\frac{1}{a'+b}}$. Then, $\delta(b) = O((a_1 - a_2)^2/b^3)$ as $b \to \infty$.

Proof. Let $x = \tau(a_1 + b, a_2 + b)$, then by definition we have

$$1 = \frac{1}{x^{a_1+b}} + \frac{1}{x^{a_2+b}} = \frac{x^{-(a_1-a_2)/2} + x^{(a_1-a_2)/2}}{x^{a'+b}}.$$

Since

$$x = 2^{\frac{1}{a'+b}} + \delta(b) = 1 + \frac{\ln 2}{a'+b} + \delta(b) + O\left(\frac{1}{(a'+b)^2}\right)$$

and

$$(1+\epsilon)^{(a_1-a_2)/2} = 1 + (a_1-a_2)\epsilon/2 + (a_1-a_2)(a_1-a_2-1)\epsilon^2/4 + O(\epsilon^3),$$

we have

$$x^{-(a_1-a_2)/2} + x^{(a_1-a_2)/2} = 2 + \frac{(a_1-a_2)^2}{2}\left(\frac{\ln 2}{a'+b} + \delta(b)\right)^2 + O\left(\left(\frac{\ln 2}{a'+b} + \delta(b)\right)^3\right).$$

We also have

$$x^{a'+b} = 2\left(1 + \delta(b)/2^{\frac{1}{a'+b}}\right)^{a'+b} = 2\left(1 + (a'+b)\delta(b)/2^{\frac{1}{a'+b}} + O(\delta(b)^2)\right).$$

By the definition of x, we have

$$\lim_{b\to\infty} \frac{(a_1-a_2)^2 \ln^2 2}{2b^2}/(2b\delta(b)) = 1.$$

This implies

$$\delta(b) = \frac{(a_1-a_2)^2 \ln^2 2}{4b^3} + o(1/b^3).$$

□

3

Gate elimination

3.1 Overview

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[95]. 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 $n \leq 3$ holds trivially. For the induction step consider an optimal circuit C computing $\mathrm{MOD}^n_{3,r}$ and its topologically minimal gate A (such a gate exists since for

$n \geq 4$, $\mathrm{MOD}_{3,r}^n$ 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: $\mathrm{MOD}_{3,0}^{n-2}$, $\mathrm{MOD}_{3,1}^{n-2}$, and $\mathrm{MOD}_{3,2}^{n-2}$ (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 degenerate function of the other input and hence can be eliminated from the circuit. The resulting circuit computes the function $\mathrm{MOD}_{3,r}^{n-1}$ so the lower bound follows by induction. The best known lower bound for $\mathrm{MOD}_{3,r}^n$ is now $2.5n - O(1)$ by Stockmeyer [102], the best known upper bound is $3n + O(1)$ by Demenkov et al. [28]. Knuth [58] (see solution to exercise 480) recently conjectured that the circuit size of $\mathrm{MOD}_{3,r}^n$ is equal to $3n - 5 - [(n + r) \bmod 3 = 0]$.

In the analysis above, we eliminated two gates by assigning $x \leftarrow 0$. If A computes, say, $xy = x \wedge y$ then we eliminate more than two gates (A becomes 0 and hence all of 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.

3.2 A $3n - o(n)$ LOWER BOUND

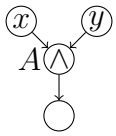
A natural idea that allows to overcome the bottleneck from the previous proof is to allow to substitute variables not only by constants but also by sums (over $\mathbb{F}_2$) of other variables. Using this idea one can prove a $3n - o(n)$ lower bound. The proof is due to Demenkov & Kulikov[29], the exposition here is due to Vadhan & Williams[106]. A function we are going to prove a lower bound for is called an affine disperser. Informally, an affine disperser is a function that cannot be made constant by sufficiently many linear substitutions (see Definition 2).

For a $3n - o(n)$ lower bound it is convenient to use xor-layered circuits. In an xor-layered circuit we allow linear sums of variables to be used as inputs to a circuit. Consider the following measure of an xor-layered 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-layered circuit C may be replaced by an input without increasing $\mu(C)$.

A $3n - 4d$ lower bound for 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-layered circuit C computing f on S, $\mu(C) \geq 4(D - d - 1)$. This can be shown by induction on D. The base case $D \leq d + 1$ is trivial. For the induction step, assume that C has the minimal value of μ. Let A be a top gate fed by linear sums x and y (such a gate must exist since f on S cannot compute a linear function for $D > d + 1$). If A computes a sum of x and y then it can be replaced by an input (without increasing μ) so assume that A computes a product, i.e., $(x \oplus c_1)(y \oplus c_2) \oplus c$ where $c_1, c_2, c \in \mathbb{F}_2$ are constants. In the fol-

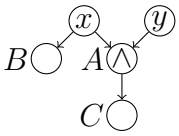
lowing we assign either $x = c_1$ or $y = c_2$. This gives us an affine subspace of $\mathbb{F}_2^n$ of dimension at least $D - 1$ (if the dimension of the resulting subspace drops to 0 this means that either x or y is constant on S contradicting the fact that the considered circuit is optimal). To proceed 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, 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 is the output gate, but this means that f is constant on an affine subspace of dimension at least d.)

Case 2. The outdegree of 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. 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.,[26] Theorem A.1) and has circuit size $n - 1$.

3.3 A $3.01n$ lower bound for affine dispersers

Following[29], we prove lower bounds for affine dispersers, that is, functions that are non-constant on affine subspaces of certain dimensions. Our first main result is the following theorem.

Theorem 2. The circuit size of an affine disperser for sublinear dimension is at least $\left(3 + \frac{1}{86}\right) n - o(n)$.

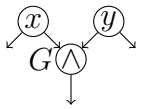
Feeding an appropriate constant to a non-linear gate (for example, AND) makes this gate constant and therefore eliminates subsequent gates, which helps to eliminate more gates than in the case of a linear gate (for example, XOR). On the other hand, linear gates, when stacked together, allow us to reorganize the circuit. This idea has been used in[83,102,14,97,29]. Then affine restrictions can kill such gates while keeping the properties of an affine disperser.

Thus, it is natural to consider a circuit as a composition of linear circuits connected by non-linear gates. 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 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[90,36,76,89] (the last reference contains an overview of previous work on cyclic circuits).

Thus we are able to make affine substitutions. We try to make such a substitution in order to make the topmost (i.e., closest to the inputs) non-linear gate constant. This, however, does not seem to be enough. The second ingredient in our proof is a *complexity measure* that manages difficult situations (bottlenecks) by allowing us to perform an amortized analysis: we count not just the number of gates, but rather a linear combination of the number of gates and the number of bottlenecks.

Our main bottleneck (called "troubled gate") is shown below.



All gates have outdegrees exactly as shown on the picture, and $\wedge$ refers to a gate computing any nonlinear operation. That is, two inputs of degree 2 feed a gate of outdegree 1 that computes $(x \oplus a)(y \oplus b) \oplus c$ where $a, b, c \in \{0, 1\}$ are

constants.

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

3.4 A $3.11n$ lower bound for quadratic dispersers

The two considered functions, MOD_3^n and an affine disperser, 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, $k \leq n-4$ and each p_i is just a variable or its negation while for affine dispersers, $k \leq n-d$ and p_i's are arbitrary linear polynomials.

A natural extension is to allow polynomials to have degree at most 2, i.e., to prove lower bounds against quadratic dispersers (see Definition 4). We prove the following result.

Theorem 3. Let $0 < \alpha \leq 1$ and $0 < \beta$ be constants satisfying

$$2^{-\frac{2+\alpha}{\beta}} + 2^{-\frac{4+2\alpha}{\beta}} \leq 1, \tag{3.1}$$

$$2^{-\frac{2}{\beta}} + 2^{-\frac{5+2\alpha}{\beta}} \leq 1, \tag{3.2}$$

$$2^{-\frac{3+3\alpha}{\beta}} + 2^{-\frac{2+2\alpha}{\beta}} \leq 1, \tag{3.3}$$

$$2^{-\frac{3}{\beta}} + 2^{-\frac{4+\alpha}{\beta}} \leq 1, \tag{3.4}$$

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

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

The constants α and β allow us to balance between 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 gates and the number of variables) 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.

For example, for an $(n, 1.83n, 2^{o(n)})$-quadratic disperser, Theorem 3 with $\alpha = 0.535$ and $\beta = 3.6513$ implies a $3.1163n - o(n) > 3.116n$ lower bound. For an $(n, 1.78n, 2^{0.03n})$-quadratic disperser, it implies a $3.006n$ lower bound.

Currently, explicit constructions of quadratic dispersers with such parameters are not known while showing their existence non-constructively is easy (see Lemma 1 and the discussion after it for the known constructions with weaker parameters). Theorem 3 can be viewed as an additional motivation for studying quadratic dispersers.

4

Lower bound of $3.01n$ for affine dispersers

4.1 Overview

In this chapter we prove a $(3 + \frac{1}{86})n - o(n)$ lower bound on the circuit size of affine dispersers.

We apply a sequence of transformations on circuits. To accomodate this we use a generalization of circuits which we call "fair semicircuits". Semicircuits may contain cycles of a certain kind; however we only introduce cycles in such a

way that the values computed in the gates are internally consistent. Section 4.2 contains the definitions, examples and properties of fair semicircuits.

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

In Section 4.3 we show how to make substitutions in fair semicircuits, and how to normalize them afterwards. We introduce five normalization rules covering various degenerate cases that may occur in a circuit after applying a substitution to it: e.g., a gate of outdegree 0, a gate computing a constant function, a gate whose value depends on one of its inputs only. For each such case, we show how to simplify the circuit.

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

Also, we define a so-called troubled gate. Informally speaking, this is a special bottleneck configuration in a circuit that does not allow us to eliminate more than three gates easily. To overcome this difficulty, we use a circuit complexity measure that depends on the number of troubled gates. This, in turn, requires us to analyze carefully how many new troubled gates can be introduced by applying a normalization rule. We show that a circuit computing an affine dis-

perser cannot have too many troubled gates (otherwise one could find an affine subspace of not too large dimension that makes the circuit constant). This implies that the bottleneck case cannot appear too often during the gate elimination process.

In Section 4.4 we formally define a source arising from constant, affine, and quadratic substitutions. We only apply simple quadratic substitutions. In particular, we maintain the following invariant: the variables from the right-hand side of quadratic substitutions are pairwise different and do not appear in the left hand side of affine substitutions. This invariant guarantees that a disperser for affine sources is also a disperser for our generalized sources (with parameters that are only slightly worse).

In Section 4.5 we define the circuit complexity measure and state the main result of this chapter: we can always reduce the measure by an appropriate amount by shrinking the space. The circuit lower bound follows from this result. The measure is defined as a linear combination of four parameters of a circuit: the number of gates, the number of troubled gates, the number of quadratic substitutions, and the number of inputs. The optimal values for coefficients in this linear combination come from solving a simple linear program.

Finally, Section 4.6 employs all developed techniques in order to prove the lower bound. Before going into details, in Section 4.6.1 we give a short outline of the case distinction argument that covers all essential cases. A complete proof is given in Section 4.6.2.

4.2 Cyclic circuits

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 vertices (inputs and 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 gates compute either $\oplus$ or $\equiv$, for technical reasons we also allow degenerate 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 gates. The operation of a gate imposes an equation defining the computed value. Whenever an input is encountered, it is treated like a constant (because we will be interested in solving this system when we are given specific input values). Thus we formally have a separate system for every assignment to the inputs; 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 example is a gate G computing $F \oplus x \oplus 1$, where x is an input; 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.

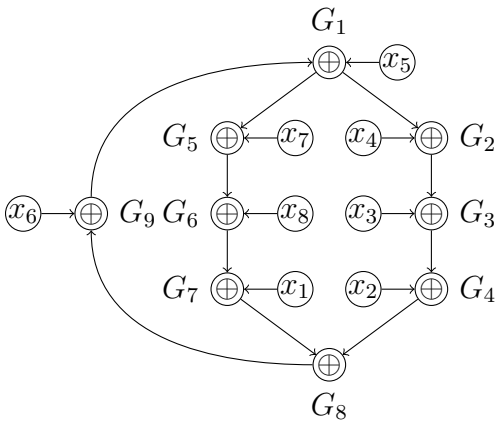
For a cyclic xor-circuit, this is a linear system with a square matrix. We call a cyclic xor-circuit *fair* if this matrix has full rank. It follows that for every as-

signment 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 inputs (clearly, it is an affine function). A simple example of a fair cyclic xor-circuit is shown in Figure 4.1. 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, degenerate gates, and constant gates).

4.2.1 Relations between 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 4.1 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 can only compute the sum of two inputs, therefore it cannot compute any output of the given function.

On the other hand, a minimal xor-circuit of k variables computing a single output has exactly $k-1$ gates and is acyclic.



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

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

Figure 4.1: A simple example of a cyclic xor-circuit. In this case all the gates are labeled with $\oplus$. The affine functions computed by the gates are shown on the right of the circuit. The bottom row shows the program computed by the circuit as well as the corresponding linear system.

4.2.2 Semicircuits

We introduce the class of *semicircuits* which is a generalization of both Boolean circuits and cyclic xor-circuits.

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

In this chapter we abuse notation by using the word "circuit" to mean a fair semicircuit.

4.3 Cyclic circuit transformations

4.3.1 Basic substitutions

In this section we consider several types of substitutions. A constant substitution to an input is straightforward:

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

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

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

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

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

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 to the system corresponding to the circuit (see Section 4.2). The transformation simply replaces every equation of the form $H = F \odot x_1$ with the equation $H = F \odot G$ (and 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 $x_2, \ldots, x_n$. Assume that the solution to the original system does not satisfy the new equation. Then take $x_1 = g(x_2, \ldots, x_n)$, it violates the corresponding equation in the original system,

a contradiction. Vice versa, consider a different solution for the new system. It must satisfy the original system (where $x_1 = g(x_2, \ldots, x_n)$), but the original system has a unique solution. □

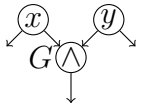
In what follows, however, we will also use substitutions that do not satisfy the hypothesis of this proposition: substitutions that create cycles. We defer this construction to Section 4.3.3.

4.3.2 Normalization and troubled gates

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 normalization rules below; however, while normalizing we need to make sure the circuit remains within certain limits: in particular, it must remain fair and compute the same function. We need to check also that we do not "spoil" a circuit by introducing "bottleneck" cases. Namely, we are going to prove an upper bound on the number of newly introduced unwanted fragments called "troubled" gates.

We say that a gate G is *troubled* if it satisfies the following three criteria:

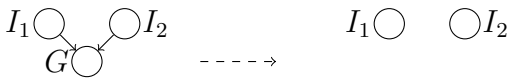
- G is an and-type gate of outdegree 1,
- the gates feeding G are inputs,
- both inputs feeding G have outdegree 2.



For simplicity, we will denote all and-type gates by $\wedge$, and all xor-type gates by $\oplus$.

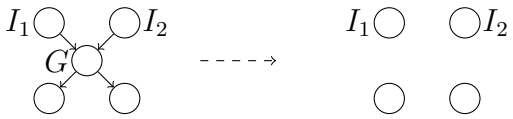
We say that a circuit is *normalized* if none of the following rules is applicable to it. Each rule eliminates a gate G whose inputs are gates I_1 and I_2. (Note that I_1 and I_2 can be inputs or gates, and, in rare cases, they can coincide with G itself.)

Rule 1: If G has no outgoing edges and is not marked as an output, then remove it.



Note also that it could not happen that the only outgoing edge of G feeds itself, because this would make a trivial equation and violate the circuit fairness.

Rule 2: If G is trivial, i.e., it computes a constant function c of the circuit inputs (not necessarily a constant operation on the two inputs of G), remove G and "embed" this constant to the next gates. That is, for every gate H fed by G, replace the operation $h(g,t)$ computed in this gate (where g is the input from G and t is the other input) by the operation $h'(g,t) = h(c,t)$. (Clearly, h' depends on at most one argument, which is not optimal, and in this case after removing G one typically applies Rule 3 or Rule 2 to its successors.)



Rule 3: If G is degenerate, i.e., it computes an operation depending only on one of its inputs, remove G by reattaching its outgoing wires to that input. This

may also require changing the operations computed at its successors (the corresponding input may be negated; note that an and-type gate (xor-type gate) remains an and-type gate (xor-type gate)).

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

If G has no outgoing edges it must be an output gate (otherwise it would be removed by Rule 1). In this special case, we remove G and mark the corresponding input of G (or its negation) as the output gate.

Rule 4: If G is a 1-gate that feeds a single gate Q, Q is distinct from G itself, and Q is also fed by one of G's inputs, then replace in Q the incoming wire going from G by a wire going from the other input of G (this might also require changing the operation at Q); then remove G. We call such a gate G *useless*.

Rule 5: If the inputs of G coincide (I_1 and I_2 refer to the same node) then we replace the binary operation $g(x, y)$ computed in G with the operation $g'(x, y) = g(x, x)$. Then perform the same operation on G as described in Rule 3 or 2.

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

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

New troubled gates. For all the rules, 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). Each of I_1, I_2 may create at most two new troubled gates. Hence each rule, when applied, introduces at most four new troubled gates. □

4.3.3 Affine substitutions

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

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

before that we need to prove a structural lemma (which is trivial for acyclic circuits) that guarantees its success.

For an xor-circuit, we say that a gate G depends on a variable x if G computes an affine function in which x is a term. Note that in a circuit without cycles this means that precisely one of the inputs of G depends on x, and one could trace this dependency all the way to x, therefore there always exists a path from x to G. In the following lemma we show that it is always possible to find such a path in a fair cyclic circuit too. However, it may be possible that some nodes on this path do not depend on x. Note that dependencies in cyclic circuits are sometimes counterintuitive. For example, in Figure 4.1, gate G_4 is fed by x_2 but does not depend on it.

Lemma 4. 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 us substitute all variables in C except for x to 0. Since G depends on x, it can only compute x or its negation.

Let $\mathcal{R}$ be the set of gates that are reachable from x, and $\mathcal{U}$ be the set of 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 a 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 $\mathcal{U}$, and thus unknowns corresponding to gates from $\mathcal{R}$ do not appear in the equations corresponding to gates from $\mathcal{U}$,
- L_U is a vector of constants, it cannot contain x since $\mathcal{U}$ is not reachable from x,
- L_R is a vector of affine functions of x, since all other inputs are substituted by zeros.

If U is singular, then the whole matrix is singular, which contradicts the fairness of C. Therefore, U is nonsingular, i.e., the values $\mathcal{G}' = (g_1, \ldots, g_{|\mathcal{U}|})^T$ are uniquely determined by $U \times \mathcal{G}' = L_U$, and they are constant (independent of x). This means that G cannot belong to $\mathcal{U}$. □

We now come to rewiring.

Lemma 5. Let C be a fair semicircuit with inputs $x_1, \ldots, x_n$ and gates $G_1, \ldots, G_m$. Let G be a gate not reachable by a directed path from any and-type gate. Assume that G computes the function $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c$, where $I \subseteq \{2, \ldots, n\}$. Let $b \in \{0, 1\}$ be a constant. Then one can transform C into a new circuit C' with the following properties:

1. graph-theoretically, C' has the same gates as C, plus a new gate Z; some edges are changed, in particular, x_1 is disconnected from the circuit;
2. the operation in G is replaced by the constant operation b;

3. $\text{in}_{C'}(Z) = 2$, $\text{out}_{C'}(G) = \text{out}_C(G)+1$, $\text{out}_{C'}(x_1) = 0$. $\text{out}_{C'}(Z) = \text{out}_C(x_1)-1$.

4. The indegrees and outdegrees of all other gates are the same in C and C'.

5. C' is fair.

6. all gates common for C' and C compute the same functions on the affine subspace defined by $x_1 \oplus \bigoplus_{i\in I} x_i \oplus c \oplus b = 0$, that is, if $f(x_1, \ldots, x_n)$ is the function computed by a gate in C and $f'(x_2, \ldots, x_n)$ is the function computed by its counterpart in C', then $f(\bigoplus_{i\in I} x_i \oplus c \oplus b, x_2, \ldots, x_n) = f'(x_2, \ldots, x_n)$. The gate Z computes the function $\bigoplus_{i\in I} x_i \oplus c \oplus b$ (which on the affine subspace equals x_1).

Proof. Consider a path from x_1 to G that is guaranteed to exist by Lemma 4. Denote the gates on this path by $G_1, \ldots, G_k = G$. Denote by $T_1, \ldots, T_k$ the other inputs of these gates. Note that we assume that $G_1, \ldots, G_k$ are pairwise different gates while some of the gates $T_1, \ldots, T_k$ may coincide with each other and with some of $G_1, \ldots, G_k$ (it might even be the case that $T_i = G_i$).

The transformation is shown in Figure 4.2. The gates $A_0, \ldots, A_k$ are shown on the picture just for convenience: any of $x_1, Z, G_1, \ldots, G_k$ may feed any number of gates, not just one A_i.

To show the fairness of C', assume the contrary, that is, the sum of a subset of rows of the new matrix is zero. The row corresponding to $G_k = b$ must belong to the sum (otherwise we would have only rows of the matrix for C, plus an extra column). However, this would mean that if we sum up the correspond-

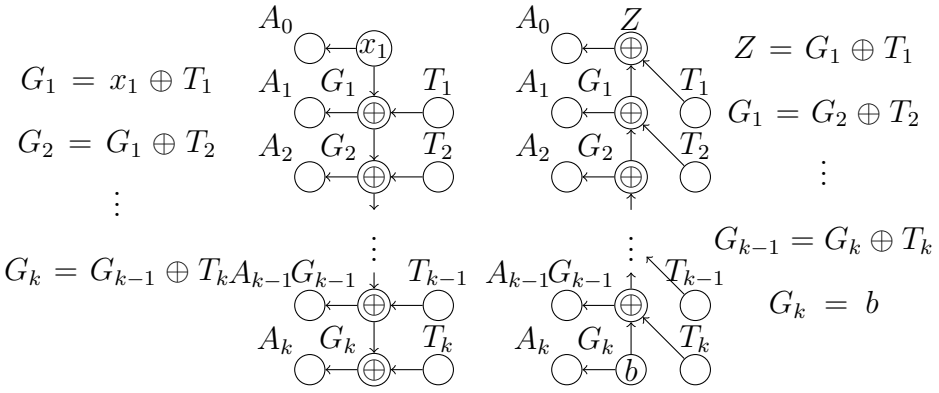


Figure 4.2: This figure illustrates the transformation from Lemma 5. We use $\oplus$ as a generic label for xor-type gates. That is, in the picture, gates labelled $\oplus$ may compute the function $\equiv$.

ing lines of the system (not just the matrix) for C, we get $G_k = \text{const} \oplus \bigoplus_{j \in J} x_j$ 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 $G_1, \ldots, G_k$ compute the same values in C' and C; the value of Z is also clearly correct. □

Corollary 1. This transformation does not introduce new troubled gates.

Proof. Indeed, the gates being fed by $G_1, \ldots, G_{k-1}, G_k, Z$ are not fed by variables; these gates themselves are not and-type gates; other gates do not change their degrees or types of inputs. □

After an application of this transformation, we apply Rule 2 to G. Since the only troubled gates introduced by this rule are the inputs of the removed gate,

no troubled gates are introduced (and one gate, G itself, is eliminated, thus the combination of Lemma 5 and Rule 2 does not increase the number of gates).

4.4 Read-once depth-2 quadratic sources

We generalize affine sources as follows.

Definition 5. Let the set of variables $\{x_1, \ldots, x_n\}$ be partitioned into three disjoint sets $F, L, Q \subseteq \{1, \ldots, n\}$ (for *free*, *linear*, and *quadratic*). Consider a system of equalities that contains

- for each variable x_j with $j \in Q$, a quadratic equality of the form

$$x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j \,,$$

  where $i, k \in F$ and c_i, c_k, c_j are constants; the variables from the right-hand side of all the quadratic substitutions are pairwise disjoint;

- for each variable x_j with $j \in L$, an affine equality of the form

$$x_j = \bigoplus_{i \in F_j \subseteq F} x_i \oplus \bigoplus_{i \in Q_j \subseteq Q} x_i \oplus c_j$$

  for 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 $d = |F|$.

An example of such a system is shown in Figure 4.3.

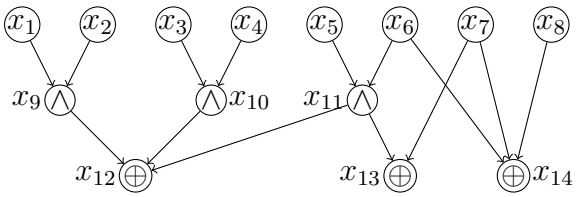


Figure 4.3: An example of an rdq-source. Note that a variable can be read just once by an and-type gate while it can be read many times by xor-type gates.

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

For this, we will gradually build a straight-line program (that is, a sequence of lines of the form $x = f(\ldots)$, where f is a function depending on the program inputs (free variables) and the values computed in the previous lines) that produces an rdq-source. We build it in a bottom-up way. Namely, we take an unprotected free variable x_j and extend our current program with either a quadratic substitution

$$x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j$$

depending on free unprotected variables x_i, x_k or a linear substitution

$$x_j = \bigoplus_{i \in J} x_i \oplus c_j$$

depending on any variables. It is clear that such a program can be rewritten into a system satisfying Definition 5. In general, we cannot use protected free variables without breaking the rdq-property. However, there are two special cases when this is possible: (1) we can substitute a constant to a protected variable (and update the quadratic line accordingly: for example, $z = xy$ and $x = 1$

yield $z = y$ and $x = 1$); (2) we can substitute one protected variable for another variable (or its negation) from the same quadratic equation (for example, $z = xy$ and $x = y$ yield $z = y$ and $x = y$).

In what follows we abuse 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 6. Let $R \subseteq \mathbb{F}_2^n$ be an rdq-source of dimension d, let the free variables be $x_1, x_2, \ldots, x_d$, and let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a function. Then f restricted to R, denoted $f|_R$, is a function $f|_R\colon \mathbb{F}_2^d \to \mathbb{F}_2$, defined by $f|_R(x_1, \ldots, x_d) = f(R(x_1, \ldots, x_d))$.

Note that affine sources are precisely rdq-sources with $Q = \emptyset$. We define dispersers for rdq-sources similarly to dispersers for affine sources.

Definition 7. 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.

The following proposition shows that affine dispersers are also rdq-dispersers with related parameters.

Proposition 4. Let $R \subseteq \mathbb{F}_2^n$ be an rdq-source of dimension d. Then R contains an affine subspace of dimension at least $d/2$.

Proof. For each quadratic substitution $x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j$, further restrict R by setting $x_i = 0$. This replaces a quadratic substitution by two affine

substitutions $x_i = 0$ and $x_j = c_i(x_k \oplus c_k) \oplus c_j$; the number of free variables is decreased by one. Also, since the free variables do not occur on 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 our quadratic substitutions are disjoint we have initially that $2|Q| \leq |F| = d$, so the number of newly introduced affine substitutions is at most $d/2$. □

Note that it is important in the proof that protected variables do not appear on the left-hand sides. 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 equations $x_1x_2 = x_3x_4 = \ldots = x_{n-1}x_n = 1$ defines a single point, so any function is constant on this set.

Corollary 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.

4.5 Circuit complexity measure

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

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

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

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

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

Lemma 6. 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, $|V| = n$. In what follows we let $\sqcup$ denote the disjoint set union. Let us call two inputs x and y neighbors if they feed the same troubled gate. 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 $v_1 + v_2 \leq n$,

$$n + 5d \leq 2t = v_1 + 2v_2 \leq n + v_2.$$

Let T be the set of inputs that feed two troubled gates, $|T| = v_2 \geq 5d$. We now

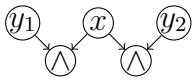
construct two disjoint subsets $X \subset T$ and $Y \subset V$ such that

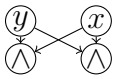
- $|X| = d$,
- there are $|Y|$ consistent linear equations that make the circuit C independent of variables from $X \sqcup Y$.

When the sets X and Y are constructed the theorem statement follows immediately. Indeed, we first take $|Y|$ equations that make C independent of $X \sqcup Y$, then we set all the remaining variables $V \setminus (X \sqcup Y)$ to arbitrary constants. After this, the circuit C evaluates to a constant (since it does not depend on variables from $X \sqcup Y$ and all other variables are set to constants). We have $|Y| + |V \setminus (X \sqcup Y)| = |V \setminus X| = n - d$ linear equations which contradicts the assumption that f is an affine disperser for dimension d.

Now we turn to constructing X and Y. For this we will repeat the following routine d times. First we pick any variable $x \in T$, it feeds two troubled gates, let y_1 and y_2 be neighbors of x (y_1 may coincide with y_2). We add x to X, also we add y_1, y_2 to Y. Note that it is possible to assign constants to y_1 and y_2 to make C independent of x. (See the figure below. If y_1 differs from y_2, then we substitute constants to them so that they eliminate troubled gates fed by x and leave C independent of x. If y_1 coincides with y_2, then either $x = c$, or $y_1 = c$, or $y_1 = x \oplus c$ eliminates both troubled gates for some constant c; if we make an $x = c$ substitution, then formally we have to interchange x and y, that is, add y rather than x to X.) Each of y_1, y_2 has at most one neighbor different from x. We remove x, y_1, y_2, neighbors of y_1 and y_2 (at most five vertices total) from

the set T, if they belong to it. Since at each step we remove at most five vertices from T, we can repeat this routine d times. Since we remove the neighbors of y_1 and y_2 from T, we guarantee that in all future steps when we pick an input, its neighbors do not belong to Y, so we can make arbitrary substitutions to them and leave the system consistent.





□

We are now ready to formulate the main bounds of this section.

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

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

and

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

We defer the proof of this theorem to Section 4.6.2. This theorem, together with Corollary 2, implies a lower bound on the circuit complexity of affine dispersers.

Corollary 3. 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 $q = 0$, $i \leq n$, $t < \frac{n}{2} + \frac{5d}{2}$ (see Lemma 6). 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 $\delta - \frac{\alpha_T}{2} - \alpha_I$ satisfying the condition from Corollary 3 is given by the following linear program: maximize $\delta - \frac{\alpha_T}{2} - \alpha_I$ subject to

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

The optimal values for this linear program are

$$\begin{aligned} \alpha_T &= \frac{1}{43}, \\ \alpha_Q &= 1 + 22\alpha_T = \frac{65}{43}, \\ \alpha_I &= 6 + 2\alpha_T = 6 + \frac{2}{43}, \end{aligned}$$

$$\begin{aligned}\beta &= 1 - 4\alpha_T = \frac{39}{43},\\ \delta &= 9 + 3\alpha_T = 9 + \frac{3}{43}.\end{aligned}$$

This gives the main result of this chapter.

Theorem 2. The circuit size of an affine disperser for sublinear dimension is at least $\left(3 + \frac{1}{86}\right) n - o(n)$.

4.6 Gate elimination

In order to prove Theorem 4 we first show that it is always possible to make a substitution and decrease the measure by δ.

Theorem 5. Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be an rdq-disperser for dimension d, let R be an rdq-source of dimension $s \geq d + 2$, and let C be an optimal (i.e., C with the smallest $\mu(C, R)$) fair semicircuit computing the function $f|_R$. Then there exist an rdq-source R' of dimension $s' < s$ and a fair semicircuit C' computing the function $f|_{R'}$ such that

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

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

Proof of Theorem 4. We prove that for optimal C computing $f|_R$, $\mu(C, R) \geq \delta(s - d - 2)$. We do it by induction on s, the dimension of R. Note that the

statement is vacuously true for $s \leq d+2$, since μ is nonnegative. 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' guaranteed to exist by Theorem 5, 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 comes from the induction hypothesis. □

4.6.1 Proof outline

The proof of Theorem 5 is based on a careful consideration of a number of cases. Before considering all of them formally in Section 4.6.2, we show a high-level picture of the case analysis.

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

We always make constant, linear or simple quadratic *substitution* to a variable. Then we remove the substituted variable from the circuit, so that for each

assignment to the remaining variables the function is defined. It is easy to make a constant substitution $x = c$ for $c \in \{0, 1\}$. We propagate the value c to the inputs fed by x and remove x from the circuit, since it does not feed any other gates. An affine substitution $x = \bigoplus_{i \in I} x_i \oplus c$ is harder to make, because a straightforward way to eliminate x would be to compute $\left(\bigoplus_{i \in I} x_i \oplus c\right)$ elsewhere. We will always have a gate G that computes $\bigoplus_{i \in I} x_i \oplus c$ and that is not reachable by a direct path from an and-type gate. Fortunately, in this case Lemma 5 shows how to compute it on the affine subspace defined by the substitution without using x and without increasing the number of gates (later, an extra gate introduced by this lemma is removed by normalization).

Thus, in this sketch we will be making arbitrary affine substitutions for sums that are computed in gates without saying that we need to run the reconstruction procedure first. Also, we will make a simple quadratic substitution $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$ only if the gates fed by z are canceled out after the substitution, so that we do not need to propagate this quadratic value to other gates. We want to stay in the class of rdq-sources, therefore we cannot make an affine substitution to a variable x if it already has been used in the right-hand side of some quadratic restriction $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$, also we cannot make quadratic substitutions that overlap in the variables. In this proof sketch we ignore these two issues, but they are addressed in the full proof in the next section.

Let A be a topologically minimal and-type gate (i.e., an and-type gate that is not reachable from any and-type gate), let I_1 and I_2 be the inputs of A (I_1

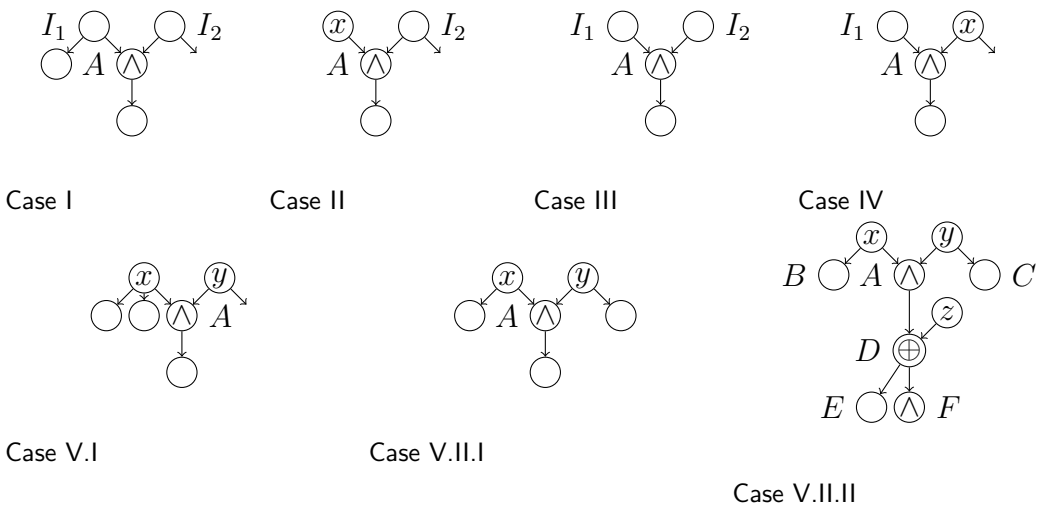


Figure 4.4: The gate elimination process in Proof Outline of Theorem 5.

and I_2 can be variables or gates). Now we consider the following cases (see Figure 4.4).

Case I. At least one of I_1, I_2 (say, I_1) is a gate of outdegree greater than one. There is a constant c such that if we assign $I_1 = c$, then A becomes constant. (For example, if A is an and, then $c = 0$, if A is an or, then $c = 1$ etc.) When A becomes constant it eliminates all the gates it feeds. Therefore, if we assign the appropriate constant to I_1, we eliminate I_1, two of the gates it feeds (including A), and also a successor of A. This is four gates total, and we decrease the measure by at least $\alpha_I + 4\beta = 9\frac{29}{43} > \delta$.

Case II. At least one of I_1, I_2 (say, I_1) is a variable of outdegree one. We assign the appropriate constant to I_2. This eliminates I_2, A, a successor of A, and I_1. This assignment eliminates at least two gates and two variables, so the measure decrease is at least $2\alpha_I + 2\beta = 13\frac{39}{43} > \delta$.

Case III. I_1 and I_2 are gates of outdegree one. Then if we assign the appropriate constant to I_1, we eliminate I_1, A, a successor of A, and I_2 (since I_2 does not feed any gates). We decrease the measure by at least $\alpha_I + 4\beta > \delta$.

Case IV. I_1 is a gate of outdegree one, I_2 is a variable of outdegree greater than one. Then we assign the appropriate constant to I_2. This assignment eliminates I_2, at least two of its successors (including A), a successor of A, and I_1 (since it does not feed any gates). Again, we decrease the measure by at least $\alpha_I + 4\beta > \delta$.

Case V. I_1 and I_2 are variables of outdegree greater than one.

Case V.I. I_1 or I_2 (say, I_1) has outdegree at least three. By assigning the appropriate constant to I_1 we eliminate at least three of the gates it feeds and a successor of A, four gates total.

Case V.II. I_1 and I_2 are variables of degree two. If A is a 2^+-gate we eliminate at least four gates by assigning I_1 so in what follows we assume that A is a 1-gate. In this case A is a troubled gate. We want to make the appropriate substitution and eliminate I_1 (or I_2), its successor, A, and A's successor.

Case V.II.I. If this substitution does not introduce new troubled gates, then we eliminate a variable, three gates and decrease the number of troubled gates by one. Thus, we decrease the measure by $\alpha_I + 3 + \alpha_T = 9\frac{3}{43} = \delta$.

Case V.II.II. If the substitution introduces troubled gates, then we consider which normalization rule introduces troubled gates. The full case analysis is presented in the next section, here we demonstrate just one case of the analysis. Let us consider the case when a new troubled gate is introduced when we eliminate the gate fed by A (see Figure 4.4, the variable z will feed a new troubled gate after assignments $x = 0$ or $y = 0$). In such a case we make a different substitution: $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$. This substitution eliminates gates A, D, E, F and a gate fed by F. Thus, we eliminate one variable, five gates, but we introduce a new quadratic substitution, and decrease the measure by at least $\alpha_I + 5\beta - \alpha_Q = 9\frac{3}{43} = \delta$.

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

4.6.2 Full proof

Proof of Theorem 5. Since normalization does not increase the measure and does not change R, we may assume that C is normalized.

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 equation or two to R.
- Since we now compute the disperser on a smaller set, we simplify C (in particular, we disconnect the substituted variables from the rest of the circuit). For this, we
  - change the operations in the gates fed by the substituted variables or restructure the xor part of the circuit according to Lemma 5,
  - apply some normalization rules to remove some gates (and disconnect substituted variables).
- We count the decrease of μ.
- We further normalize the circuit (without increase of μ) to bring it to the normalized state required for the next induction step.

Since $s \geq d + 2$, even if we add two more lines 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 the measure after subsequent normalization).

1. Perform two consecutive affine substitutions to reduce the number of influential inputs by at least three. 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 at least 2: $\Delta\mu \geq 2\alpha_I$ (numerically, this case is subsumed by the previous one).

3. Perform one affine substitution to kill four gates: $\Delta\mu \geq 4\beta + \alpha_I$.

4. Perform one constant substitution to eliminate three gates including at least one troubled gate so that no new troubled gate is introduced: $\Delta\mu \geq \alpha_I + 3 + \alpha_T$.

5. Perform one *quadratic* substitution to kill five gates: $\Delta\mu \geq 5\beta - \alpha_Q + \alpha_I$.

6. Perform two affine substitutions to kill at least five gates and replace a quadratic substitution by an affine one, reducing the measure by at least $5\beta + \alpha_Q + 2\alpha_I$. By substitution this is $\Delta\mu \geq 2.5\beta + \frac{\alpha_Q}{2} + \alpha_I$.

7. Perform one affine substitution to kill two gates and replace one quadratic substitution by an affine one: $\Delta\mu \geq 2\beta + \alpha_Q + \alpha_I$.

All substitutions that we perform are of the form such that adding them to an rdq-source results in a new rdq-source.

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 normalization rules where applicable.

Note that the measure can accidentally drop less than we expect if new troubled gates emerge. We take care of this when counting the number of gates that disappear. In particular, recall Proposition 3 that guarantees the decrease of

β per one eliminated gate. If some additional gate accidentally disappears, it may introduces new troubled gates but does not increase the measure, because $\beta \geq 0$.

4.6.3 Cases:

Case 1. The circuit contains a protected variable q that either feeds an and-type gate or feeds at least two gates. Then there is a type 7 substitution of q by a constant.

Case 2. The circuit contains a protected 0-variable q occurring in the right-hand side of a quadratic substitution together with some variable q'. We substitute a constant to q'. After this neither q nor q' are influential, so we have a type 2 substitution.

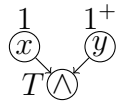
Note that after this case all protected variables are 1-variables feeding xor gates.

Case 3. The circuit contains a variable x feeding an and-type gate T, and $out(x) + out(T) \geq 4$. Then if x gets the value that trivializes T, we remove four gates: T by Rule 2, and descendants of x and T by Rule 3. If some of these descendants coincide, this gate becomes trivial (instead of degenerate) and is removed by Rule 2 (instead of Rule 3), and an additional gate (a descendant of this descendant) is removed by Rule 3. This makes a type 3 substitution.

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

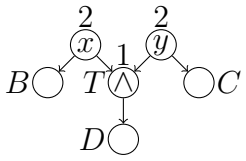
or two.

Case 4. There is an and-type gate T fed by two inputs x and y, one of which (say, x) has outdegree 1. Adopt the notation from the following picture. In this and all the subsequent pictures we show the outdegrees near the gates that are important for the case analysis.



We substitute y by a 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 inputs x and y, and at this point (due to the cases 3 and 4) we inevitably have $out(T) = 1$ and $out(x) = out(y) = 2$, that is, T is "troubled". Adopt the notation from the following picture:



Since the circuit is normalized, $B \neq D$ and $C \neq D$ (Rule 4). One can now remove three gates by substituting a constant to x that trivializes T. If in addition to the three gates one more gate can be killed, we are done (substitution of type 3). Otherwise, we have just three gates, but

the troubled gate T is removed. If this does not introduce a new troubled gate, it makes a substitution of type 4. Likewise, if this is the case for a substitution to y, we are done.

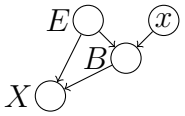
So in the remaining subcases of Case 5 we will be fighting the situation where only three gates are eliminated while one or more troubled gates are introduced.

How can it happen that a new troubled gate is introduced? This means that something has happened around some and-type gate E. Whatever has happened, it is due to two gates, B and D, that became degenerate (if some of them became trivial, then one more gate would be removed). The options are:

- *E gets as input a variable* instead of a gate (because some gate in between became degenerate).
- A variable *increases its outdegree* from 1 to 2 (because a gate of degree at least two became degenerate), and this variable starts to feed E (note that it could not feed it before, because after the increase it would feed it twice).
- A variable *decreases its outdegree* to 2. This variable could not feed E before this happens, because this would be Case 3. It takes at least one degenerate gate, X, to pass a new variable to E, thus the decrease of the outdegree has happened because of a single degenerate gate Y. In order to decrease the outdegree of the variable this

gate must have outdegree 1, thus it would be removed by Rule 4 as useless.

- *E decreases its outdegree* to 1.
  - This could happen if two gates, B and D, became degenerate, and they fed a single gate. However, in this case E should already have 2-variables as its inputs, Case 3.
  - This could also happen if E feeds B and some gate X, and B becomes degenerate to X. However, in this case B is useless (Rule 4). (Note that $out(B) = 1$, because otherwise E would not decrease its outdegree to 1.)



  - Similarly, if E feeds D and some gate X, and D becomes degenerate to X.

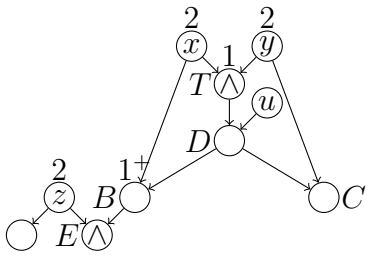
Summarizing, only the two first possibilities could happen, and both pass some variable to E through either B or D (or both).

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. If $B = C$, then one can trivialize both T and B either by substituting a constant to x or y or by one affine substitution $y = x \oplus c$

(using 2) for the appropriate constant c (this can be easily seen by examining the possible operations in the two gates). Since x and y are unprotected, the number of influential variables is decreased by 2, making a substitution of type 2.

Case 5.2. Assume that D feeds both B and C. In this case, a new troubled gate may emerge only because D is fed by a variable u, and it is passed to some and-type gate E. Note that $out(D) \leq 2$, 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.

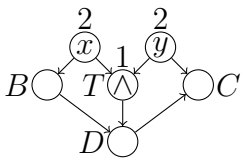


If $out(B) \geq 2$, then even if $out(u) = 1$, it must be that $C = E$ or that B feeds C, because otherwise u would become a 3-variable after substituting x. Neither is possible: $C = E$ would imply $B = D$ and $y = z$, contradicting the assumption that $D \neq B$ (from the assumption of Case 5); if B feeds C, that means that $B = D$, which is impossible. Therefore, we conclude that $out(B) = 1$. So 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.

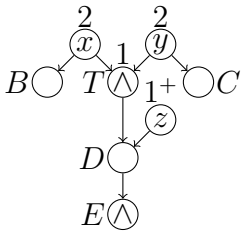
Case 5.3. Assume now that B feeds D, and D feeds C. (Or, symmetrically, C feeds D, and D feeds B.) Then substituting y to trivialize T removes T, D, and C. Now we show that this substitution introduces *no new troubled gates*, which contradicts our assumption about new troubled gates. The gates C and D are degenerate the 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.



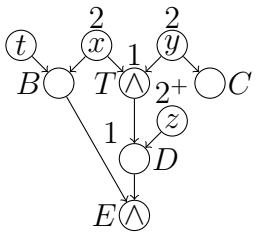
Case 5.4. We can now assume that B and D are not connected (in any direction).

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. Assume that 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).



- If $out(z) \geq 2$, then $out(D) = 1$ and E is fed by another variable t either directly or via B. In the former case, we can substitute t to trivialize E, this kills E and the gate it feeds, and also makes D and then T 0-gates; a type 3 substitution. In the latter case:



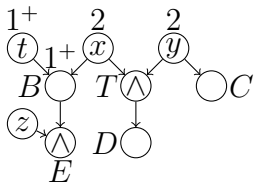
  - if $out(B) \geq 2$, then B is a xor-type gate (see Case 3), and by substituting $x = t \oplus c$ for the appropriate constant c, we can make B a constant trivializing E and remove two more descendants of B and E, a type 3 substitution;
  - if $out(B) = 1$, then we can set z and y to constants trivializing 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.

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.

- If D is an and-type gate, substituting z by the appropriate constant trivializes D and kills both gates that it feeds; also T becomes a 0-gate, a type 3 substitution.
- If z is protected, we set x and z to constants trivializing T, D, and E. This additionally removes B and the gates that E feeds, at least five gates in total. Since we also kill a quadratic substitution, this makes a type 6 substitution.
- Since we can now assume that z is unprotected and D is an xor-type gate, we can make a substitution $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$ for appropriate constants c_1, c_2, c_3 to assign D a value that trivializes E. This makes T a 0-gate and removes also D, E, another gate that D feeds, and the gate(s) that E feeds. As usual, if some degenerate gates coincide, another gate is removed. Taking into account the penalty for introducing a quadratic substitution, 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).



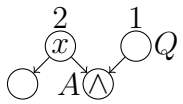
- If $out(B) \geq 2$ (which means B is a xor-type gate, see Case 3), then by substituting $x = t \oplus c$ (using Proposition 2) for the appropriate constant c, we can make B a constant trivializing E and remove two more descendants of B and E, a type 3 substitution.
- If $out(B) = 1$, then we can set 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 (otherwise it computes an affine function, and a single affine substitution makes it

constant). 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 inputs.

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



Recall that x is unprotected due to Case 1, and x cannot feed Q because of Rule 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 Rule 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 a gate Q. Denote the other input by P. If P is also a 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. 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 Rule 2. For this, we use the xor-reconstruction procedure described in Lemma 5. In order to perform it, we need at least one unprotected variable x_i with $i \in I$.

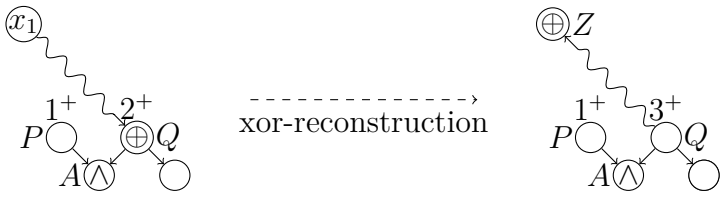
Case 7.1. Such a variable x_1 exists.

We then add the substitution $x_1 = b \oplus c \oplus \bigoplus_{i \in I \setminus \{1\}} x_i$ 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 5. Note that it only changes the in- and outdegrees of x_1 (replacing it by a new gate Z) and Q. No new troubled gates are introduced, and the subsequent application of Rule 2 to Q removes Q without introducing new troubled gates as well.

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

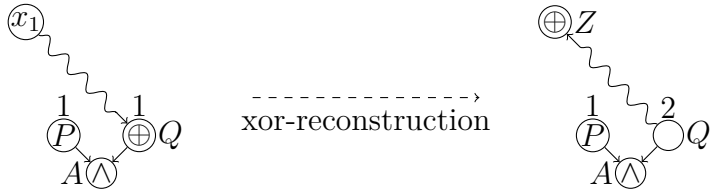
Below we go through several subcases depending on the type of the gate P.

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



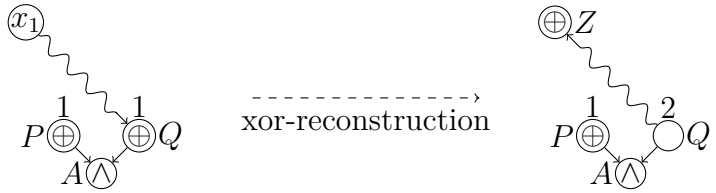
After the reconstruction, there are at least three descendants of Q and at least one descendant of A, a type 3 substitution.

Case 7.1.2. Q is a 1-gate and P is an input. 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. This means that P is left untouched by the xor-reconstruction. After trivializing A the circuit becomes independent of both x_1 and P giving a type 2 substitution.

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



Again, P is left untouched by the xor-reconstruction since it only has one successor and it is of and-type while the xor-reconstruction is performed in the linear part of the circuit. After the substitution, we remove two successors of Q, at least one successor of A, and make P a 0-gate. A type 3 substitution. Note that P cannot be a successor of Q because of Rule 4.

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

Case 7.2.1. Both inputs to Q, say x_j and x_k, are variables, and they occur in the same quadratic substitution $w = (x_j \oplus c)(x_k \oplus c') \oplus c''$. Then perform a substitution $x_j = x_k \oplus c'''$ (using Proposition 2) in order to trivialize the gate A. It kills the quadratic substitution (and does not harm other quadratic substitutions, because x_j and x_k could not occur in them), Q, A, its descendant (and more, but we do not need it), 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 substitution $x_p = (x_j \oplus a)(x_k \oplus b) \oplus c$. Recall that at this point all protected variables are 1-variables feeding xor-gates (see Cases 1 and 2). We substitute x_k by a constant d and normalize the circuit. This eliminates the successor of x_k, kills the quadratic substitution, 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 xor-gate Q' (if it were an input 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. This kills four gates. 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 xor-type* 1*-gate.*

Case 7.2.3. P is an input. Then it has the following properties as in Case 7.1.2. Take any $j \in I$ and assume that x_j appears with x_k in a quadratic substitution. 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 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. Both inputs to P or Q (say, P) are variables x_p and x_q. Let x_j be a variable from the affine function computed at Q and let x_k be its couple. Note that $x_j \neq x_p, x_q$ while it might be the case that $x_k = x_p$ or $x_k = x_q$. We substitute x_k by a constant to make x_j unprotected. We then trivialize A by an affine substitution to x_j. This way, we kill the dependence on three variables by two substitutions. A type 1 substitution. *Thus in what follows we can assume that both P and Q have at least one xor-gate as an input.*

Case 7.2.4.2. 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. We substitute x_k by a constant and normalize the descendant of x_k. It only kills one xor-gate fed by x_k and makes x_j unprotected. Note that at this point P is still a 1-xor. We then trivialize A by substituting x_j by an affine function. Similarly to Case 7.1.3, this kills four gates and gives, for two substitutions, $\Delta\mu \geq 5\beta + \alpha_Q + 2\alpha_I$. A type 6 substitution.

Case 7.2.4.3. Since P and Q, and gates that feed them all compute non-trivial functions (because of Rule 2), the only case when the condition of the previous case does not apply is the following: P computes an affine function on a single variable x_i, Q com-

putes an affine function on a single variable x_j, the variables x_i and x_j appear together in a quadratic substitution, and moreover x_i feeds Q while x_j feeds P. But this is just impossible. Indeed, since x_i is a protected variable it only feeds Q. As Q computes an affine function on x_i, Lemma 4 guarantees that there is a path from x_i to Q. But this path must go through P and A leading to a cycle that goes through an and-type gate A.

□

5

Lower bound of $3.11n$ for quadratic dispersers

5.1 Overview

In this chapter we introduce the weighted gate elimination method. This method allows us to give a simple proof of a $3.11n$ lower bound for quadratic dispersers against xor-layered circuits. We define xor-layered circuits as a generalization of Boolean circuits in Section 5.2. Section 5.3 defines weighted gate elimination and proves the lower bound. We note that there are no known ex-

plicit constructions of quadratic dispersers with the parameters needed for our proof, and refer the reader to Section 2.2 for the known constructions with weaker parameters.

We prove this lower bound by extending the gate elimination method. The proof goes by induction on the size of the quadratic variety S on which the circuit computes the original function correctly. Note that for affine varieties, after k substitutions we have $|S| = 2^{n-k}$, while for quadratic varieties this relation no longer holds. (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.) 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.

5.2 Preliminaries

By an *xor-layered* circuit we mean a circuit whose inputs may be labeled not only by input variables but also by sums of variables. One can get an xor-layered circuit from a regular circuit by replacing xor-gates that depend on two inputs by an input (see Figure 5.1).

We will need the following technical lemma.

Lemma 7. Let $0 < \alpha \leq 1$ and $0 < \beta$ be constants satisfying inequalities (3.4), (3.1):

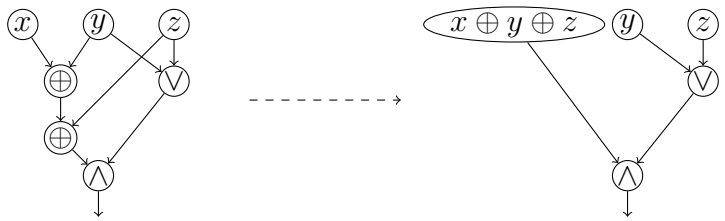


Figure 5.1: An example of a transformation from a regular circuit to an xor-layered circuit.

$$2^{-\frac{3}{\beta}} + 2^{-\frac{4+\alpha}{\beta}} \leq 1,$$
$$2^{-\frac{2+\alpha}{\beta}} + 2^{-\frac{4+2\alpha}{\beta}} \leq 1.$$

Then

$$2^{-\frac{4}{\beta}} + 2^{-\frac{4}{\beta}} \leq 1, \tag{5.1}$$
$$2^{-\frac{3+\alpha}{\beta}} + 2^{-\frac{3+2\alpha}{\beta}} \leq 1. \tag{5.2}$$

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

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

In order to prove the inequality (5.2), we use Heinz's inequality[47]:

$$\frac{x^{1-t}y^t + x^t y^{1-t}}{2} \leq \frac{x+y}{2} \text{ for } x, y > 0, 0 \leq t \leq 1.$$

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

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

□

In this chapter we abuse notation by using the word "circuit" to mean an xor-layered circuit.

5.3 Weighted Gate Elimination

The main result of this chapter is the following theorem.

Theorem 3. Let $0 < \alpha \leq 1$ and $0 < \beta$ be constants satisfying

$$2^{-\frac{2+\alpha}{\beta}} + 2^{-\frac{4+2\alpha}{\beta}} \leq 1, \tag{3.1}$$
$$2^{-\frac{2}{\beta}} + 2^{-\frac{5+2\alpha}{\beta}} \leq 1, \tag{3.2}$$
$$2^{-\frac{3+3\alpha}{\beta}} + 2^{-\frac{2+2\alpha}{\beta}} \leq 1, \tag{3.3}$$
$$2^{-\frac{3}{\beta}} + 2^{-\frac{4+\alpha}{\beta}} \leq 1, \tag{3.4}$$

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

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

As noted in Section 3.4, this theorem implies a lower bound $3.11n$ for $(n, 1.83n, 2^{o(n)})$-quadratic dispersers, and a lower bound $3.006n$ for $(n, 1.78n, 2^{0.03n})$-quadratic dispersers.

In the next lemma, we use the following circuit complexity measure: $\mu(C) = G(C)+\alpha\cdot I(C)$ where $0 < \alpha \leq 1$ is a constant to be determined later. Theorem 3 follows from this lemma with $S = \mathbb{F}_2^n$, which is an $(n, 0)$-quadratic variety.

Lemma 8. Let $f \in B_n$ be an (n, k, s)-quadratic disperser, $S \subseteq \mathbb{F}_2^n$ be an (n, t)-quadratic variety, $0 < \alpha \leq 1, 0 < \beta$ be constants satisfying inequalities (3.1), (3.2), (3.3), (3.4), C be an xor-layered circuit that computes f on S. Then

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

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

To prove the induction step we proceed as follows. If $t \geq k$ 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 $\mu(C) - \mu(C_i) \geq \Delta_i$. Then, by the induction hypothesis,

$$\mu(C) \geq \mu(C_i) + \Delta_i \geq$$

$$\min\{\beta(\log_2|S_i| - \log_2 s - 1), 2(k-(t+1))\} + \Delta_i =$$
$$\min\{\beta(\log_2|S| - \log_2 s - 1) + (\Delta_i + \beta\log_2 p_i), 2(k-t) + (\Delta_i - 2)\}.$$

Hence, if $\Delta_i \geq -\beta\log_2 p_i$ and $\Delta_i \geq 2$ for either $i = 0$ or $i = 1$ then the required inequality follows by the induction hypothesis. The inequality $\Delta_i \geq -\beta\log_2 p_i$ is true whenever $p_i \geq 2^{-\frac{\Delta_i}{\beta}}$. 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

$$2^{-\frac{\Delta_0}{\beta}} + 2^{-\frac{\Delta_1}{\beta}} \leq 1 \text{ and } \Delta_0, \Delta_1 \geq 2. \tag{5.3}$$

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 (5.3). For this, we use the inequalities (3.1)–(3.4),(5.1)–(5.2).

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 $\max\{|S_0|, |S_1|\} \geq |S|/2 > s$ which contradicts the fact that f is an (n, k, s)-quadratic disperser.

Let A be an and-gate with the maximal number of and-gates on a path from A 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-layered 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

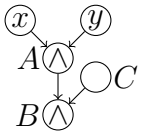
feed A.

Case 1. $\text{outdeg}(x) = \text{outdeg}(y) = 1$.

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

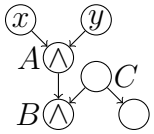
Let C be the other input of B (it might be an input as well as a non-input gate).

Case 1.1.1. $\text{outdeg}(C) = 1$.



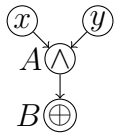
We make A constant. Then the gate B is eliminated. Moreover, either $A = 0$ or $A = 1$ trivializes the gate B so all its successors and the gate C are also eliminated (since C is only used to compute B, but B now computes a constant). In both cases x and y are not needed anymore (as the only gate A that was fed by both these inputs is now constant). So, we get $\{\Delta_0, \Delta_1\} = \{2 + 2\alpha, 3 + 3\alpha\}$. (Or $\{2 + 2\alpha, 4 + 2\alpha\}$, but it is even better as $\alpha \leq 1$, which we use in the rest of the analysis without further mentioning it.) The required inequalities (5.3) follows from (3.3).

Case 1.1.2. $\text{outdeg}(C) \geq 2$.



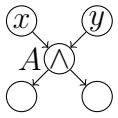
Because of the choice of A, the gate C computes a polynomial of degree at most 2. We make C constant. In both cases we eliminate two successors of C and C itself. This reduces the measure by at least $2+\alpha$. In one of the cases B is trivialized which causes the removal of the successors of B, the 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 (5.3) because of (3.1).

Case 1.2. outdeg$(A) = 1$ 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 (5.3) are satisfied due to (5.2).

Case 1.3. outdeg$(A) \geq 2$.



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 (5.2) imply that (5.3) holds.

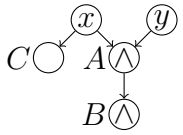
Case 2. Outdegree of either x or y is at least 2. Say, $\text{outdeg}(x) \geq 2$.

Case 2.1. $\text{outdeg}(A) = 1$ 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. $\text{outdeg}(y) = 1$.



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 (5.2) guarantees that (5.3) holds.

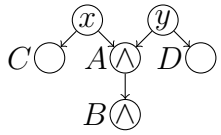
Case 2.1.1.2. B is trivialized when $A = c \oplus 1$.

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 (5.3) are satisfied because of (3.1).

Case 2.1.2. $\text{outdeg}(y) \geq 2$.

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



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

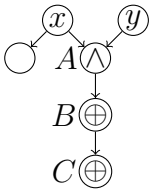
If $A = c$ we eliminate A, B, and the successors of B. If $A = c \oplus 1$ we eliminate A, B, C, D, x, and y. If $C = D$ then this gate becomes constant so all its successors are also eliminated. Hence $\{\Delta_0, \Delta_1\} = \{3, 4+2\alpha\}$. The inequalities (5.3) are satisfied because (3.4).

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. In this case we need to take additional care to show that we eliminate five gates even if some of the mentioned five gates coincide. 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 (3.2) ensures (5.3).

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

Case 2.2.1. outdeg$(B) = 1$ and B feeds an xor-gate C.

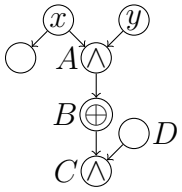


Because of the choice of A, we know that the gate C computes a quadratic polynomial. We make C constant. In both cases we eliminate A, B, C, and the successors of C. Hence $\Delta_0 = \Delta_1 = 4$. The inequalities (5.3) are satisfied because of (5.1).

Case 2.2.2. outdeg$(B) = 1$ and B feeds an and-gate C.

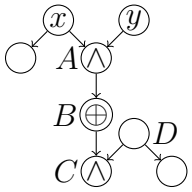
Let D be the other input of C. Note that if $D = A$ then the circuit is not optimal (C depends on A and the other input of B so one can compute C directly without using B).

Case 2.2.2.1. outdeg$(D) = 1$.



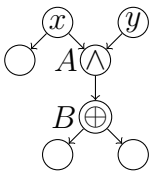
We make B constant. In both cases we eliminate A, B, and C. Moreover, when B is the constant trivializing C we eliminate also D and the successors of C. The gate D contributes (to the complexity decrease) $\alpha \leq 1$ if it is an input gate and 1 if it is not an input. Hence we have $\{\Delta_0, \Delta_1\} = \{3, 4 + \alpha\}$. The inequality (3.4) guarantees that (5.3) is satisfied.

Case 2.2.2.2. outdeg$(D) \geq 2$.



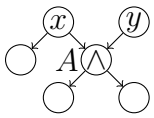
We make D constant (we are allowed to do so because it computes a polynomial of degree at most 2). In both cases we eliminate D and its successors and reduce the measure by at least $2 + \alpha$ (as D might be an input). In the case when C becomes constant we eliminate also the successors of C as well 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 (5.3) are satisfied because (3.1) and $\alpha \leq 1$.

Case 2.2.3. outdeg$(B) \geq 2$.



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 (5.3) are satisfied because of (5.1).

Case 2.3. outdeg$(A) \geq 2$.



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, $\{\Delta_0, \Delta_1\} = \{3, 4 + 2\alpha\}$. The inequality (3.4) implies that (5.3) is satisfied.

□

6

Circuit SAT Algorithms

6.1 Overview

The most efficient known *algorithms* for the #SAT problem on binary Boolean circuits use similar case analyses to the ones in gate elimination. Chen and Kabanets[20] recently showed that the known case analyses can also be used to prove *average case circuit lower bounds*, that is, lower bounds on the size of approximations of an explicit function.

In this chapter, we provide a general framework for proving worst/average case lower bounds for circuits and upper bounds for #SAT that is built on ideas of Chen and Kabanets. A proof in such a framework goes as follows. One starts

by fixing three parameters: a class of circuits, a circuit complexity measure, and a set of allowed substitutions. The main ingredient of a proof goes as follows: by going through a number of cases, one shows that for any circuit from the given class, one can find an allowed substitution such that the given measure of the circuit reduces by a sufficient amount. This case analysis immediately implies an upper bound for #SAT. To obtain worst/average case circuit complexity lower bounds one needs to present an explicit construction of a function that is a disperser/extractor for the class of sources defined by the set of substitutions under consideration. Then the worst-case circuit lower bound can be obtained by gate elimination, and the average-case circuit lower bound follows from Azuma-type inequalities for supermartingales.

We show that many known proofs (of circuit size lower bounds and upper bounds for #SAT) fall into this framework. Using this framework, we prove the following new bounds: average case lower bounds of $3.24n$ and $2.59n$ for circuits over U_2 and B_2, respectively (though the lower bound for the basis B_2 is given for a quadratic disperser whose explicit construction is not currently known), and faster than 2^n #SAT-algorithms for circuits over U_2 and B_2 of size at most $3.24n$ and $2.99n$, respectively. Recall that by B_2 we mean the set of all bivariate Boolean functions, and by U_2 the set of all bivariate Boolean functions except for parity and its complement.

6.1.1 New results

The main qualitative contribution of this chapter is a general framework for

proving circuit worst/average case lower bounds and #SAT upper bounds. This framework is separated into conceptual and technical parts. The conceptual part is a proof that for a given circuit complexity measure and a set of allowed substitutions, for any circuit, there is a substitution that reduces the complexity of the circuit by a sufficient amount. This is usually shown by analyzing the structure of the top of a circuit. The technical part is a set of lemmas that allows us to derive worst/average case circuit size lower bounds and #SAT upper bounds as one-line corollaries from the corresponding conceptual part. The technical part can be used in a black-box way: given a proof that reduces the complexity measure of a circuit (conceptual part), the technical part implies circuit lower bounds and #SAT upper bounds. For example, by plugging in the proofs by Schnorr and by Demenkov and Kulikov, one immediately gets the bounds given by Chen and Kabanets. We also give new proofs that lead to the quantitatively better results.

The main quantitative contribution of this chapter is the following new bounds which are currently the strongest known bounds:

- average case lower bounds of $3.24n$ and $2.59n$ for circuits over U_2 and B_2 (though the lower bound for the basis B_2 is given for a quadratic disperser whose explicit construction is not currently known), respectively, improving upon the bounds of $2.99n$ and $2.49n$[20];
- faster than 2^n #SAT-algorithms for circuits over U_2 and B_2 of size at most $3.24n$ and $2.99n$, respectively, improving upon the bounds of $2.99n$ and $2.49n$[20].

6.1.2 Framework

We prove circuit lower bounds (both in the worst case and in the average case) and upper bounds for #SAT using the following four step framework.

Initial setting We start by specifying the three main parameters: a class of circuits $\mathcal{C}$, a set $\mathcal{S}$ of allowed substitutions, and a circuit complexity measure μ. A set of allowed substitutions naturally defines a class of "sources". For the circuit lower bounds we consider functions that are non-constant (dispersers) or close to uniform (extractors) on corresponding sets of sources. In this chapter we focus on the following four sets of substitutions where each set extends the previous one:

1. Bit fixing substitutions, $\{x_i \leftarrow c\}$: substitute variables by constants.
2. Projections, $\{x_i \leftarrow c, x_i \leftarrow x_j \oplus c\}$: substitute variables by constants and other variables and their negations.
3. Affine substitutions, $\{x_i \leftarrow \bigoplus_{j \in J} x_j \oplus c\}$: substitute variables by affine functions of other variables.
4. Quadratic substitutions, $\{x_i \leftarrow p\colon \deg(p) \leq 2\}$: substitute variables by degree two polynomials of other variables.

Case analysis We then prove the main technical result stating that for any circuit from the class $\mathcal{C}$ there exists (and can be constructed efficiently) an allowed substitution $x_i \leftarrow f \in \mathcal{S}$ such that the measure μ is reduced by a sufficient amount under both substitutions $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$.

#SAT upper bounds As an immediate consequence, we obtain an upper bound on the running time of an algorithm solving #SAT for circuits from $\mathcal{C}$. The corresponding algorithm takes as input a circuit, branches into two cases $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$, and proceeds recursively. When applying a substitution $x_i \leftarrow f \oplus c$, it replaces all occurrences of x_i by a subcircuit computing $f \oplus c$. The case analysis provides an upper bound on the size of the resulting recursion tree.

Circuit size lower bounds Then, by taking a function that survives under sufficiently many allowed substitutions, we obtain lower bounds on the average case and worst case circuit complexity of the function. Below, we describe such functions, i.e., dispersers and extractors for the classes of sources under consideration.

1. The class of bit fixing substitutions generates the class of *bit-fixing sources*[25]. Extractors for bit-fixing sources find many applications in cryptography (see[33] for an excellent survey of the topic). The standard function that is a good disperser and extractor for such sources is the parity function $x_1 \oplus \cdots \oplus x_n$.
2. Projections define the class of *projection sources*[82]. Dispersers for projections are used to prove lower bounds for depth-three circuits[82]. It is shown[82] that a binary BCH code with appropriate parameters is a disperser for $n - o(n)$ substitutions. See[84] for an example of extractor with good parameters for projection sources.

3. Affine substitutions give rise to the class of *affine sources*. There are several known constructions of dispersers[12,98] and extractors[117,67,11,68] that are resistant to $n - o(n)$ substitutions.

4. The class of quadratic substitutions generates a special case of *polynomial sources*[35,11] and *quadratic varieties sources*[34]. Although an explicit construction of a function resistant to sufficiently many quadratic substitutions* is not currently known, it is easy to show that a random function is resistant to any $n - o(n)$ quadratic substitutions.

6.2 Preliminaries

Following the approach from[20], we use a variant of Azuma's inequality with one-sided boundedness condition in order to obtain average case lower bounds. The standard version of Azuma's inequality requires the difference between two consecutive variables to be bounded, and[20] considers the case when the difference takes on only two values but is bounded only from one side. For our results, we need a slightly more general variant of the inequality: the difference between two consecutive variables takes on up to k values and is bounded from one side. We give a proof of this inequality, which is an adjustment of proofs from[71,3,20].

A sequence $X_0, \ldots, X_m$ of random variables is a *supermartingale* if for every $0 \leq i < m$, $\mathrm{E}[X_{i+1}|X_i, \ldots, X_0] \leq X_i$.

*We note that a disperser for quadratic substitutions is a weaker object than a quadratic disperser defined in Section 5, and thus might be easier to construct.

Lemma 9. Let $X_0, \ldots, X_m$ be a supermartingale, let $Y_i = X_i - X_{i-1}$. If $Y_i \leq c$ and for fixed values of $(X_0, \ldots, X_{i-1})$, the random variable Y_i is distributed uniformly over at most $k \geq 2$ (not necessarily distinct) values, then for every $\lambda \geq 0$:

$$\Pr[X_m - X_0 \geq \lambda] \leq \exp\left(\frac{-\lambda^2}{2mc^2(k-1)^2}\right).$$

Note that we have an extra factor of $(k-1)^2$ comparing to the usual form of Azuma's inequality, but we do not assume that $X_i - X_{i-1}$ is bounded from below.

Proof. For any $t > 0$,

$$\begin{aligned}\Pr\left[X_m - X_0 \geq \lambda\right] &= \Pr\left[\sum_{i=1}^{m} Y_i \geq \lambda\right] = \Pr\left[\exp\left(t \cdot \sum_{i=1}^{m} Y_i\right) \geq e^{\lambda t}\right] \\ &\leq e^{-\lambda t} \cdot \mathrm{E}\left[\exp\left(t \cdot \sum_{i=1}^{m} Y_i\right)\right].\end{aligned}$$

First we show that for any $t > 0$, $\mathrm{E}[e^{tY_i}] \leq \exp\left(t^2c^2(k-1)^2/2\right)$. Since $\{X_i\}$ is a supermartingale, $\mathrm{E}[Y_i | X_{i-1}, \ldots, X_0] \leq 0$. W.l.o.g., assume that $\mathrm{E}[Y_i | X_{i-1}, \ldots, X_0] = 0$, otherwise we can increase the values of negative Y_i's which only increases the objective function $\mathrm{E}[e^{tY_i}]$. Note that $\mathrm{E}[Y_i] = 0, Y_i \leq c$ and Y being uniform over k values imply that $|Y_i| \leq c(k-1)$. Let

$$h(y) = \frac{e^{tc(k-1)} + e^{-tc(k-1)}}{2} + \frac{e^{tc(k-1)} - e^{-tc(k-1)}}{2} \cdot \frac{y}{c(k-1)}$$

be the line going through points $\left(-c(k-1), e^{-tc(k-1)}\right)$ and $\left(c(k-1), e^{tc(k-1)}\right)$.

From convexity of e^{tY}, $e^{tY} \leq h(y)$ for $|y| \leq c(k-1)$. Thus,

$$\mathrm{E}[e^{tY_i}] \leq \mathrm{E}[h(Y_i)] = h\left(\mathrm{E}[Y_i]\right) = h(0) = \cosh(tc(k-1)) \leq \exp\left(t^2c^2(k-1)^2/2\right),$$

where the last inequality $\cosh(x) \leq \exp\left(x^2/2\right)$ for $x > 0$ can be proven by comparing the Taylor series of the two functions.

Now,

$$\mathrm{E}\left[\exp\left(t \cdot \sum_{i=1}^{m} Y_i\right)\right] = \mathrm{E}\left[\exp\left(t \cdot \sum_{i=1}^{m-1} Y_i\right) \cdot \mathrm{E}\left[\exp\left(t \cdot Y_m\right) | X_{m-1}, \ldots, X_0\right]\right] \leq$$
$$\mathrm{E}\left[\exp\left(t \cdot \sum_{i=1}^{m-1} Y_i\right)\right] \cdot \exp\left(t^2c^2(k-1)^2/2\right) \leq \exp\left(mt^2c^2(k-1)^2/2\right),$$

which for $t = \lambda/mc^2(k-1)^2$ implies $\Pr[X_m - X_0 \geq \lambda] \leq \exp\left(\frac{-\lambda^2}{2mc^2(k-1)^2}\right)$. □

6.3 Main theorem

In this section we prove the main technical theorem that allows us to get circuit complexity lower bounds and #SAT upper bounds.

Definition 8. Let $\{v_1, \ldots, v_m\}$ be splitting vectors, and each v_i is a splitting vector of length $2^{t_i} \geq 2$. For a class of circuits Ω (e.g., $\Omega = B_2$ or $\Omega = U_2$), a set of substitutions $\mathcal{S}$, and a circuit complexity measure μ, we write

$$\mathrm{Splitting}(\Omega, \mathcal{S}, \mu) \preceq \{v_1, \ldots, v_m\}$$

as a shortcut for the following statement: For any normalized circuit C from the

class Ω one can find in time poly($|C|$) either a substitution[†] from $\mathcal{S}$ whose splitting vector with respect to μ belongs to the set $\{v_1, \ldots, v_m\}$ or a substitution that trivializes the output gate of C. A substitution always trivializes at least one gate (in particular, when we assign a constant to a variable we trivialize an input gate) and eliminates at least one variable.

Theorem 6. If Splitting$(\Omega, \mathcal{S}, \mu) \preceq \{v_1, \ldots, v_m\}$ and the longest splitting vector has length 2^k, then[‡]

1. There exists an algorithm solving #SAT for circuits over Ω in time $O^*(\gamma^{\mu(C)})$, where
$$\gamma = \max_{i \in [m]} \{\tau(v_i)\}\,.$$

2. If $f \in B_n$ is an $(\mathcal{S}, n, r)$-disperser, then
$$\mu(f) \geq \beta_w \cdot (r - k + 1)\,, \text{ where } \beta_w = \min_{i \in [m]} \{\overline{v_i}_{\max}\}\,.$$

3. If $f \in B_n$ is an $(\mathcal{S}, n, r, \epsilon)$-extractor, then for every $\mu < \beta_a \cdot r$,
$$\mu\,(f, \delta) \geq \mu, \text{ where } \beta_a = \min_{i \in [m]} \left\{\overline{v_i}_{\text{avg}}\right\} \text{ and } \beta_m = \min_{i \in [m]} \{\overline{v_i}_{\min}\}\,,$$
$$\delta = \epsilon + \exp\left(\frac{-(r \cdot \beta_a - \mu)^2}{2r(\beta_a - \beta_m)^2(2^k - 1)^2}\right)\,.$$

Proof. We present a proof for a special case when all splitting vectors have

[†]Here we assume that the circuit obtained from C by the substitution and normalization belongs to Ω too.

[‡]See Section 2.4 for the definitions of $\overline{v}_{\max}, \overline{v}_{\min}$, and $\overline{v}_{\text{avg}}$.

length 2 (i.e., $k = 1$): $\{v_1, \ldots, v_m\} = \{(a_1, b_1), \ldots, (a_m, b_m)\}$. This makes the exposition simpler, and it is easy to see that the general statement follows by the same argument. In this case,

$$\beta_w = \min_{i \in [m]}\{\max\{a_i, b_i\}\}, \beta_a = \min_{i \in [m]} \left\{ \frac{a_i + b_i}{2} \right\}, \beta_m = \min_{i \in [m]}\{\min\{a_i, b_i\}\}.$$

1. Consider the following branching algorithm for #SAT. We describe the algorithm as a branching tree, each node of which contains a Boolean circuit and a set of currently made substitutions. The root of the tree is $(C, \emptyset)$ — the input circuit and an empty set of substitutions. The nodes where the circuit is trivialized are called leaves. At each internal node (a node that is not a leaf) the algorithm finds in polynomial time substitutions $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$ guaranteed by the theorem statement. Then the algorithm recursively calls itself on two circuits obtained from the current one by substituting $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$. That is, the algorithm adds to the current node (C, S) two children $(C|x_i \leftarrow f, S \cup \{x_i \leftarrow f\})$ and $(C|x_i \leftarrow f \oplus 1, S \cup \{x_i \leftarrow f \oplus 1\})$. Note that the statement guarantees that the substitutions $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$ either give us an (a_i, b_i)-splitting for some i (i.e., decrease the measure μ by at least a_i in one branch, and b_i in the other one), or trivialize the circuit and produce two leaves.

   At each leaf the algorithm counts the number V of satisfying assignments: If the formula is constant zero, then $V = 0$, otherwise, $V = 2^v$, where v is the number of variables in the current formula. Indeed, for each assign-

ment to the v variables, there exists a unique assignment to the rest of the variables (via the substitutions at the leaf), and the circuit remains constant 1. Since substitutions $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$ lead to different assignments to the input variables, the leaves of the branching tree correspond to disjoint sets of assignments. Therefore, the total number of satisfying assignments of the original circuit is the sum of the number of satisfying assignments at the leaves of the tree. Since the running time of the algorithm at each node is polynomial, the total running time is bounded from above by $O^*\left(\gamma^{\mu(C)}\right)$, where $\gamma = \max_{i \in [m]}\{\tau(a_i, b_i)\}$.

2. For every pair of integers (n, r) such that $n \geq r \geq 0$, let $\mathcal{F}_{n,r} \subseteq B_n$ denote the class of functions from $\{0,1\}^n$ to $\{0,1\}$ that are not constant after any r substitutions from $\mathcal{S}$. We show that for every $f \in \mathcal{F}_{n,r}$, $\mu(f) \geq \beta_w \cdot (r - k + 1)$.

   The proof of the claim proceeds by induction on r. For $r < k$ the statement is trivial. Now assume that $r \geq k$. Consider substitutions $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$ guaranteed by the lemma statement. Now select a value of $c \in \{0,1\}$ in such a way that the substitution $x_i \leftarrow f \oplus c$ reduces the measure by at least β_w. Consider the function g of $n-1$ variables which is f restricted to $x_i \leftarrow f \oplus c$. By the theorem statement, $\mu(f) \geq \beta_w + \mu(g)$, and by the induction hypothesis, $\mu(g) \geq \beta_w \cdot (r - 1 - k + 1)$. Therefore, $\mu(f) \geq \beta_w \cdot (r - k + 1)$.

3. Let us consider a circuit C such that $\mu(C) \leq \beta_a \cdot r$. Consider the branching

tree from the 1st part of the proof. At each node of the branching tree let us uniformly at random choose a child we proceed to. Let δ_i be the random variable that equals to the measure decrease at ith step (ith level of the branching tree, where 0 corresponds to the root). For $i \geq 0$, define the random variable

$$X_i = (i+1) \cdot \beta_a - \sum_{j=0}^{i} \delta_j.$$

Let us show that the variable X_i is a supermartingale:

$$\begin{aligned} \mathrm{E}[X_i | X_{i-1}, \ldots, X_0] &= i \cdot \beta_a - \sum_{j=0}^{i-1} \delta_j + (\beta_a - \mathrm{E}[\delta_i | X_{i-1}, \ldots, X_0]) \\ &= X_{i-1} + (\beta_a - \mathrm{E}[\delta_i]) \leq X_{i-1}. \end{aligned}$$

Let $Y_i = X_i - X_{i-1}$. Then Y_i is distributed uniformly over at most 2^k values, and $Y_i \leq \beta_a - \beta_m$. Now let $\lambda = \beta_a \cdot r - \mu(C)$. Then, by Lemma 9:

$$\Pr[X_r - X_0 \geq \lambda] \leq \exp\left(\frac{-\lambda^2}{2r(\beta_a - \beta_m)^2(2^k - 1)^2}\right).$$

Now we want to bound from above the correlation between f and the function given by the branching tree. Note that all leaves of the tree that have depth smaller than r altogether give correlation at most ϵ with the extractor f (since each of these leaves defines an $(\mathcal{S}, n, r)$ source). Now let us count the number of leaves at the depth at least r. There are at most 2^r possible leaves, but each of them survives till the rth level only with probability $\Pr[X_r - X_0 \geq \lambda]$. Indeed, if $X_r - X_0 < \lambda$, then $\sum_{j=1}^{r} \delta_j > \mu(C)$,

which means that the function becomes constant before the rth level. Therefore, there are at most $2^r \cdot \Pr[X_r - X_0 \geq \lambda]$ leaves at the depth at least r. Since each leaf at the depth r has r inputs fixed, it covers at most 2^{n-r} points of the Boolean cube. Therefore, the total correlation is bounded from above by:

$$\begin{aligned} \mathrm{Cor}(f, C) &\leq \epsilon + \exp\left(\frac{-\lambda^2}{2r(\beta_a - \beta_m)^2(2^k - 1)^2}\right) \\ &= \epsilon + \exp\left(\frac{-(r \cdot \beta_a - \mu(C))^2}{2r(\beta_a - \beta_m)^2(2^k - 1)^2}\right). \end{aligned}$$

□

6.4 Bounds for the basis U_2

In this and the following sections, we give proofs of the known and new circuit lower bounds and upper bounds for #SAT using the described framework. The main ingredient of all proofs is the case analysis showing the existence of a substitution reducing the measure by a sufficient amount. Usually, in such proofs we argue as follows: take a gate A and make a substitution trivializing A; this eliminates A and all its successors. However it might be the case that A is the output gate and so does not have any successors. This means that we are at the end of the gate elimination process or at the leaf of a recursion tree. This, in turn, means that we do not need to estimate the current measure decrease. For this reason, in all the proofs below we assume implicitly that if we trivialize a gate then it is not the output gate.

6.4.1 Bit fixing substitutions

We start with a well-known case analysis of a $3n-3$ lower bound for the parity function over U_2 due to Schnorr[95]. Using this case analysis we reprove the bounds given recently by Chen and Kabanets[20] in our framework. The analysis is basically the same, although the measure is slightly different. We provide these results mostly as a simple illustration of usage of the framework.

Lemma 10. $\text{Splitting}(U_2, \{x_i \leftarrow c\}, s+\alpha i) \preceq \{(\alpha, 2\alpha), (3+\alpha, 3+\alpha), (2+\alpha, 4+\alpha)\}$.

Proof. Let A be a top-gate (that is, a gate fed by two inputs) computing $(x_i \oplus a)(x_j \oplus b) \oplus c$ where x_i, x_j are input variables and $a, b, c \in \{0,1\}$ are constants. If $\text{out}(x_i) = \text{out}(x_j) = 1$ we split on x_i. When $x_i \leftarrow a$ the gate A trivializes and the resulting circuit becomes independent of x_j. This gives $(\alpha, 2\alpha)$. Assume now that $\text{out}(x_i) \geq 2$. Denote by B the other successor of x_i and let C, D be successors of A, B, respectively. Note that $B \neq C$ since the circuit is normalized but it might be the case that $C = D$. We then split on x_i. Both A and B trivialize in at least one of the branches and their successors are also eliminated. This gives us either $(3+\alpha, 3+\alpha)$ or $(2+\alpha, 4+\alpha)$. (Note if A and B trivialize in the same branch and $C = D$ then we counted C twice in the analysis above. However in this case C also trivializes so all its successors are also eliminated.) □

Corollary 4. 1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over U_2 of size at most $(3-\epsilon)n$ can be solved in time $(2-\delta)^n$.

2. $C_{U_2}(x_1 \oplus \cdots \oplus x_n \oplus c) \geq 3n - 6$.[§]

3. $C_{U_2}\left(x_1 \oplus \cdots \oplus x_n \oplus c, \exp\left(\frac{-(t-9)^2}{18(n-1)}\right)\right) \geq 3n-t$. This, in particular, implies that $\mathrm{Cor}(x_1 \oplus \cdots \oplus x_n \oplus c, C)$ is negligible for any circuit C of size $3n - \omega(\sqrt{n \log n})$.

Proof. 1. First note that for large enough α, we have $\tau(\alpha, 2\alpha) < \tau(3+\alpha, 3+\alpha) = 2^{\frac{1}{3+\alpha}} < \tau(2+\alpha, 4+\alpha)$. Let $\gamma(\alpha) = \tau(2+\alpha, 4+\alpha) - 2^{\frac{1}{3+\alpha}}$. By Lemma 3, $\gamma(\alpha) = O(1/\alpha^3)$ holds. The running time of the algorithm is at most

$$\begin{aligned}(\tau(2+\alpha, 4+\alpha))^{s+\alpha n} &\leq \left(2^{\frac{1}{3+\alpha}}(1+\gamma(\alpha))\right)^{s+\alpha n} \leq 2^{\frac{s+\alpha n}{3+\alpha}} 2^{(s+\alpha n)\gamma(\alpha)\log_2 e} \\ &\leq 2^{\frac{(3-\epsilon)n+\alpha n}{3+\alpha} + O(n/\alpha^2)} \leq (2-\delta)^n\end{aligned}$$

for some $\delta > 0$ if we set $\alpha = c/\epsilon$ for large enough $c > 0$.

2. The parity function takes a uniform random value after any $n-1$ substitutions of variables to constants. Lemma 10 guarantees that for $\alpha = 3$ we can always assign a constant to a variable so that $s + 3i$ is reduced by at least 6. Hence for any circuit C over U_2 computing parity, $s(C) + 3n \geq 6(n-1)$ implying $s(C) \geq 3n - 6$.

3. Let us consider a circuit C of size at most $3n-t$, that is, $\mu(C) \leq (3n-t)+\alpha n$. Now we fix $\alpha = 6$, then $\beta_a = \min\{9, 9, 9\} = 9, \beta_m = \min\{6, 9, 8\} = 6$.

We use the third item of Theorem 6 with $k = 1$, $r = n-1$, $\epsilon = 0$, $\mu =$

[§](We include this item only for completeness. In fact, a simple case analysis shows that $C_{U_2}(x_1 \oplus \cdots \oplus x_n) = 3n - 3$.

$(3n - t + 6n)$, which gives us

$$\delta = \exp\left(\frac{-(9(n-1) - (3n - t + 6n))^2}{18(n-1)}\right) = \exp\left(\frac{-(t-9)^2}{18(n-1)}\right).$$

□

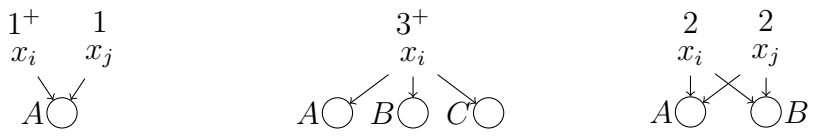
6.4.2 Projection substitutions

In this subsection, we prove new bounds for the basis U_2. The two main ideas leading to improved bounds are using projections to handle the Case 3 below and using 1-variables to get better estimates for complexity decrease (this trick was used by Zwick[118] and then by[65,53]).

Lemma 11. For $0 \leq \sigma \leq 1/2$,

$$\text{Splitting}(U_2, \{x_i \leftarrow c, x_i \leftarrow x_j \oplus c\}, s + \alpha i - \sigma i_1) \preceq$$
$$\{(\alpha, 2\alpha), (2\alpha, 2\alpha, 2\alpha, 3\alpha), (3 + \alpha + \sigma, 3 + \alpha), (4 + \alpha + \sigma, 2 + \alpha)\}.$$

Proof. Note that for every eliminated gate we decrease the measure by at least $1 - \sigma \geq 1/2$, if some gate becomes constant we decrease the measure by at least 1.



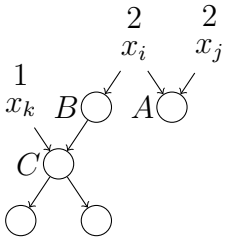
Case 1 Case 2 Case 3

Case 1. There is a top gate A fed by 1-variable x_j. Assigning x_i a constant we trivialize the gate A in one of the branches and lose the dependence on x_j. Thus, we get at least $(\alpha, 2\alpha)$ splitting.

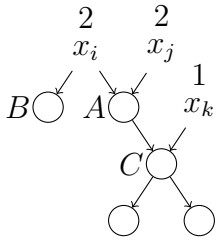
Case 2. There is a variable x_i of degree at least 3. Neither of A, B, C is fed by a 1-variable otherwise we would be in the Case 1. When we assign x_i a constant, gates A, B, and C become constant in one of the branches. Hence in one of the branches we eliminate also at least one extra gate. Thus, we get at least $(4 + \alpha - \sigma, 3 + \alpha)$ splitting vector which dominates $(3 + \alpha + \sigma, 3 + \alpha)$ (since $\sigma \leq 1/2$).

Case 3. There are two 2-variables that feed the same two top gates. Let the gates A and B compute Boolean functions $f_A(x_i, x_j) = (x_i \oplus a_A)(x_j \oplus b_A) \oplus c_A$ and $f_B(x_i, x_j) = (x_i \oplus a_B)(x_j \oplus b_B) \oplus c_B$ respectively. If $a_A = a_B$ or $b_A = b_B$ then we assign $x_i \leftarrow a_A$ or $x_j \leftarrow b_A$ respectively and make both gates constant. Otherwise, $f_B(x_i, x_j) = (x_i \oplus a_A \oplus 1)(x_j \oplus b_A \oplus 1) \oplus c_B$. It is easy to see that if $x_i \oplus a_A \oplus x_j \oplus b_A = 1$ then both functions are constant. Hence, the substitution $x_i \leftarrow a_A \oplus x_j \oplus b_A \oplus 1$ makes A and B constant as well. In both cases there is a substitution that makes A and B constant and therefore eliminates the dependence on x_j, so we get at least $(\alpha, 2\alpha)$ splitting vector.

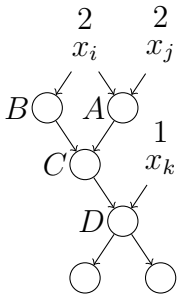
Case 4. There are three gates that are fed by two 2-variables.



Case 4.1



Case 4.2



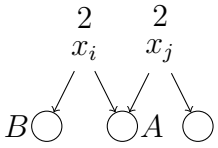
Case 4.3

Case 4.1. Gate B is a 1-gate with a successor C that is a 2^+-gate fed by the 1-variable x_k. If we assign constants to x_j and x_k we eliminate the dependence on x_i as well in one of the branches, so we get $(2\alpha, 2\alpha, 2\alpha, 3\alpha)$ splitting.

Case 4.2. Gate A is a 1-gate with a successor C that is a 2^+-gate fed by 1-variable x_k. Analogous to the previous case if we assign constants to x_i and x_j we eliminate the dependence on x_k in one of the branches, so we get again $(2\alpha, 2\alpha, 2\alpha, 3\alpha)$ splitting.

Case 4.3. Gates A and B and its common successor C are 1-gates and the only successor of C is a 2-gate fed by 1-variable x_k. When we assign constants to x_k gate D becomes constant in one of the branches, hence, gates A, B, C become unnecessary and we lose the dependence on variable x_i. So we have at least $(\alpha, 2\alpha)$ splitting.

Case 4.4. None of three previous cases apply.



Previous cases ruled out the possibility that A or B has the only successor that contributes only $1-\sigma$ to the measure decrease: we know that each of A and B is either a 2^+-gate and its successors contribute at least $2(1-\sigma) \geq 1$ or a 1-gate with a successor which is not a 2^+-gate fed by a 1-variable and it contributes at least 1. We also know, that when A and B are 1-gates with a common successor, this successor is not a 2^+-gate fed by a 1-variable, and hence it contributes at least 1.

Therefore, if we assign x_i a constant each of A and B becomes constant in one of the branches, so the successors of A and B either contribute at least 1 in both branches or contribute at least 2 in one of the branches. In addition, x_j becomes a 1-variable in the branch where A trivializes. Thus, we get either $(3+\alpha+\sigma, 3+\alpha)$ or $(4+\alpha+\sigma, 2+\alpha)$.

□

Corollary 5. 1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over U_2 of size at most $(3.25-\epsilon)n$ can be solved in time $(2-\delta)^n$.

2. Let $f \in B_n$ be an $\left(n, r(n) = n - \log^{O(1)}(n)\right)$-projections disperser from[68]. Then $C_{U_2}(f) \geq 3.5n - \log^{O(1)}(n)$.

3. Let $f \in B_n$ be an $\left(n, r(n) = n - \sqrt{n}, \epsilon(n) = 2^{-n^{\Omega(1)}}\right)$-projections extractor from[84]. Then $C_{U_2}(f, \delta) \geq 3.25n - t$, where $\delta = 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(t-10.25\sqrt{n})^2}{190.125(n-\sqrt{n})}\right)$.

This, in particular, implies that $\mathrm{Cor}(f, C)$ is negligible for any circuit C of size $3.25n - \omega(\sqrt{n \log n})$.

Proof. 1. Let $\sigma = 1/2$. First note that for large enough α, we have

$$\begin{aligned} \tau(\alpha, 2\alpha) &< \tau(2\alpha, 2\alpha, 2\alpha, 3\alpha) &< \tau(3.25 + \alpha, 3.25 + \alpha) = 2^{\frac{1}{3.25+\alpha}} \\ &< \tau(3.5 + \alpha, 3 + \alpha) &< \tau(4.5 + \alpha, 2 + \alpha). \end{aligned}$$

Let $\gamma(\alpha) = \tau(4.5+\alpha, 2+\alpha) - 2^{\frac{1}{3.25+\alpha}}$. By Lemma 3, $\gamma(\alpha) = O(1/\alpha^3)$ holds. The running time of the algorithm is at most

$$\begin{aligned} (\tau(4.5 + \alpha, 2 + \alpha))^{s+\alpha n} &\leq \left(2^{\frac{1}{3.25+\alpha}}(1 + \gamma(\alpha))\right)^{s+\alpha n} \leq 2^{\frac{s+\alpha n}{3.25+\alpha}} 2^{(s+\alpha n)\gamma(\alpha)\log_2 e} \\ &\leq 2^{\frac{(3.25-\epsilon)n+\alpha n}{3.25+\alpha} + O(n/\alpha^2)} \leq (2 - \delta)^n \end{aligned}$$

for some $\delta > 0$ if we set $\alpha = c/\epsilon$ for large enough $c > 0$.

2. Lemma 11 guarantees that for $\alpha = 7$, $\sigma = 0.5$ one can always make an affine substitution reducing $s + 7i$ by at least 10.5. The function f is resistant to $r(n)$ such substitutions. Hence for a circuit C computing f, $s(C) + 7n \geq 10.5r(n)$.

3. Let us consider a circuit C of size at most $3.25n - t$, that is, $\mu(C) \leq (3.25n - t) + \alpha n$. Now we fix $\alpha = 7$, $\sigma = 0.5$, then $\beta_a = \min\{10.5, 15.75, 10.25, 10.25\} = 10.25$, $\beta_m = \min\{7, 7, 10, 9\} = 7$.

   We use the third item of Theorem 6 with $k = 2$, $r = n - \sqrt{n}$, $\epsilon = 2^{-n^{\Omega(1)}}$,

$\mu = (3.25n - t + 7n)$, which gives us

$$\begin{aligned}\delta &= 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(10.25(n-\sqrt{n}) - (10.25n - t))^2}{2(n-\sqrt{n}) \cdot 3.25^2 \cdot 3^2}\right) \\ &= 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(t - 10.25\sqrt{n})^2}{190.125(n-\sqrt{n})}\right).\end{aligned}$$

□

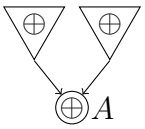
6.5 Bounds for the basis B_2

6.5.1 Affine substitutions

Here, we again start by reproving the bounds for B_2 by Chen and Kabanets[20] by using the case analysis due to Demenkov and Kulikov[29].

Lemma 12. Splitting$(B_2, \{x_i \leftarrow \oplus_{j \in J} x_j \oplus c\}, \mu = s + \alpha i) \preceq \{(\alpha, 2\alpha), (2 + \alpha, 3 + \alpha)\}$.

Proof. Fix any topological ordering of a given circuit C and let A be the first gate in this ordering which is not a 1-xor (if there is no such gate then all the gates in C are 1-xors hence C computes an affine function and we can trivialize it with a single affine substitution). Note that the subcircuit underneath the gate A is a tree of xors, that is, a subcircuit consisting of 1-xors only. Let P and Q be inputs to A. Each of P and Q is computed by a tree of xors. Since each gate in such a tree has outdegree 1, it is not used in any other part of the circuit. Also, both P and Q might as well be input gates.



In any case, we can trivialize, say, P by an affine substitution. If P is an input gate this can be done simply by assigning the corresponding variable a constant. If P is an internal gate then it computes a sum $\bigoplus_{j\in J} x_j \oplus c$. To trivialize it, we select any variable $i \in J$ and make a substitution $x_i \leftarrow \bigoplus_{j\in J\setminus\{i\}} x_j \oplus c'$. This clearly makes P constant. To remove x_i from the circuit we replace the whole tree for P by a new tree computing $\bigoplus_{j\in J\setminus\{i\}} x_j \oplus c'$ (at this point, we use essentially the fact that all the gates in the tree computing P were needed to compute P only; hence when P is trivialized all these gates may be removed safely). We then replace all occurrences of x_i by this new tree. The new tree has one gate less than the old one. So when P is an internal gate, by trivializing it we eliminate a variable and the gate P itself.

Case 1. A is a 2^+-xor. Then A itself is computed by a tree of 1-xors. Trivializing it gives $(3+\alpha, 3+\alpha)$.

Case 2. A is an and-gate and one of its inputs (say, P) is an internal gate. We trivialize P. In both branches we eliminate A and P, but in one of them A is trivialized so we eliminate also its successors. This gives $(2+\alpha, 3+\alpha)$.

Case 3. A is an and-gate fed by two variables x_i and x_j.

Case 3.1. The outdegree of one of them (say, x_i) is at least 2. Then splitting on x_i gives $(2+\alpha, 3+\alpha)$.

Case 3.2. $\mathrm{out}(x_i) = \mathrm{out}(x_j) = 1$. Then splitting on x_i is $(\alpha, 2\alpha)$.

□

Corollary 6. 1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over B_2 of size at most $(2.5 - \epsilon)n$ can be solved in time $(2 - \delta)^n$.

2. Let $f \in B_n$ be an $\left(n, r(n) = n - \log^{O(1)}(n)\right)$-affine disperser from[68]. Then $C_{B_2}(f) \geq 3n - \log^{O(1)}(n)$.

3. Let $f \in B_n$ be an $\left(n, r(n) = n - O(n/\log\log n), \epsilon(n) = 2^{-n^{\Omega(1)}}\right)$-affine extractor from[67]. Then $C_{B_2}(f, \delta) \geq 2.5n - t$, where $\delta = 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(t - O(n/\log\log n))^2}{O(n)}\right)$. This, in particular, implies that $\mathrm{Cor}(f, C)$ is negligible for any circuit C of size $2.5n - \omega(n/\log\log n)$.

Proof. 1. First note that for large enough α, we have

$$\tau(\alpha, 2\alpha) < \tau(2.5 + \alpha, 2.5 + \alpha) = 2^{\frac{1}{2.5+\alpha}} < \tau(2 + \alpha, 3 + \alpha).$$

Let $\gamma(\alpha) = \tau(2 + \alpha, 3 + \alpha) - 2^{\frac{1}{2.5+\alpha}}$. By Lemma 3, $\gamma(\alpha) = O(1/\alpha^3)$ holds. The running time of the algorithm is at most

$$\begin{aligned}(\tau(2 + \alpha, 3 + \alpha))^{s+\alpha n} &\leq \left(2^{\frac{1}{2.5+\alpha}}(1 + \gamma(\alpha))\right)^{s+\alpha n} \leq 2^{\frac{s+\alpha n}{2.5+\alpha}} 2^{(s+\alpha n)\gamma(\alpha)\log_2 e} \\ &\leq 2^{\frac{(2.5-\epsilon)n+\alpha n}{2.5+\alpha} + O(n/\alpha^2)} \leq (2 - \delta)^n\end{aligned}$$

for some $\delta > 0$ if we set $\alpha = c/\epsilon$ for large enough $c > 0$.

2. Lemma 12 guarantees that for $\alpha = 3$ one can always make an affine substitution reducing $s + 3i$ by at least 6. The function f is resistant to $r(n)$ such substitutions. Hence for a circuit C computing f, $s(C) + 3n \geq 6r(n)$.

3. Let us consider a circuit C of size at most $2.5n - t$, that is, $\mu(C) \leq (2.5n - t) + \alpha n$. Now we fix $\alpha = 5$, then $\beta_a = \min\{7.5, 7.5\} = 7.5, \beta_m = \min\{5, 7\} = 5$.

   We use the third item of Theorem 6 with $k = 1$, $r = r(n)$, $\epsilon = \epsilon(n)$, $\mu = (2.5n - t + 5n)$, which gives us

$$\begin{aligned} \delta &= \epsilon(n) + \exp\left(\frac{-(7.5r(n) - (7.5n - t))^2}{12.5r(n)}\right) \\ &= \epsilon(n) + \exp\left(\frac{-(t - 7.5(n - r(n)))^2}{12.5r(n)}\right). \end{aligned}$$

□

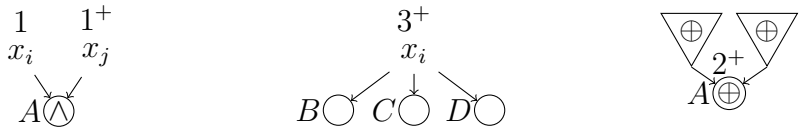
6.5.2 Quadratic substitutions

Lemma 13. For $0 \leq \sigma \leq 1/5$,

$$\text{Splitting}(B_2, \{x_i \leftarrow p\colon \deg(p) \leq 2\}, s + \alpha i - \sigma i_1) \preceq$$
$$\{(\alpha, 2\alpha), (2\alpha, 2\alpha, 2\alpha, 3\alpha), (3 + \alpha - 2\sigma, 3 + \alpha - 2\sigma), (3 + \alpha + \sigma, 2 + \alpha)\}.$$

Proof. Fix any topological order of a given circuit C and let A be the first gate in this ordering which is not a 1-xor (if there is no such gate then all the gates in C are 1-xors hence C computes an affine function and we can trivial-

ize it with a single affine substitution). Then each input of A is a tree of xors, that is, a subcircuit consisting of 1-xors only. When we do an affine substitution to some variable that feed an xor-tree, we rebuild the tree and reduce the number of gates in it by at least one (it is explained in details in the proof of Lemma 12).



Case 1 Case 2 Case 3

(In all the pictures of this proof we show only the type of the gates but not the actual functions computed at them.)

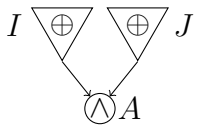
Case 1. A is a top and-gate fed by a 1-variable x_i. Similarly to the Case 1 of Lemma 11 we get $(\alpha, 2\alpha)$.

Case 2. There is a variable x_i of degree at least 3. Neither of B, C, D is an and-gate fed by a 1-variable otherwise we would be in the Case 1. If, say, B is an xor 2^+-gate fed by the 1-variable x_k we can trivialize it by an affine substitution $x_i \leftarrow x_k \oplus c$ and eliminate two variables in both branches, so we get $(2\alpha, 2\alpha)$. Otherwise we assign x_i a constant and eliminate three gates in every branch, all the gates contribute 1 to the measure decrease. Thus, we get at least $(3 + \alpha, 3 + \alpha)$ which dominates $(3 + \alpha - 2\sigma, 3 + \alpha - 2\sigma)$.

Case 3. A is 2^+-xor. Let A compute $c_A \oplus \bigoplus_{i \in I} x_i$ and $I \subseteq \{1, \ldots, n\}$, $|I| \geq 2$. If all x_i, $i \in I$, are 1-variables then for any $j \in I$ a substitution $x_j \leftarrow c \oplus$

$\bigoplus_{i\in I\setminus\{j\}} x_i$ eliminates the dependence on at least one 1-variable, so we get $(2\alpha, 2\alpha)$. Otherwise, there is at least one 2-variable x_i, $i \in I$. Substituting $x_i \leftarrow c \oplus \bigoplus_{k\in I\setminus\{i\}} x_k$ we eliminate three gates in both branches, so we get at least $(3 + \alpha - 2\sigma, 3 + \alpha - 2\sigma)$.

Case 4. A is an and-gate which is not a top gate.

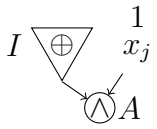


Let $I, J \subset \{1, \ldots, n\}$ be the sets of indices of variables in the left and in the right xor-trees respectively. W.l.o.g., we assume that $|I| > 1$, i.e. there is at least one gate in the left xor-tree feeding A.

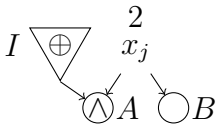
Case 4.1. There is a 1-variable x_i in the left tree. An affine substitution $x_j \leftarrow c \oplus \bigoplus_{k\in J\setminus\{j\}} x_k$ for some $j \in J$ eliminates two variables in one of the branches: the variable x_i becomes unnecessary in the branch where A becomes constant. The splitting is at least $(\alpha, 2\alpha)$.

Case 4.2. There is at least one gate in the right tree, i.e. $|J| \geq 2$. We apply an affine substitution $x_i \leftarrow c \oplus \bigoplus_{k\in I\setminus\{i\}} x_k$ for some $i \in I$ and eliminate four gates in one branch and two in the other one. This gives $(4 + \alpha - \sigma, 2 + \alpha)$ which dominates $(3 + \alpha + \sigma, 2 + \alpha)$.

Case 4.3. The right tree consists of only one variable x_j.



Case 4.3.1



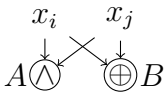
Case 4.3.2

Case 4.3.1. x_j is a 1-variable. An affine substitution $x_i \leftarrow c \oplus \bigoplus_{k \in I \setminus \{i\}} x_k$ for some $i \in I$ eliminates x_j in the branch where A becomes constant, so we get at least $(\alpha, 2\alpha)$.

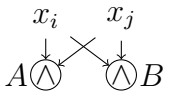
Case 4.3.2. x_j is a 2-variable. Assigning x_j a constant we eliminate four gates in one branch and two in the other one. Gate B does not introduce new an 1-variable: if B is a 2^+-gate fed by 1-variable we would be either in the Case 1 or in the Case 3. The splitting on x_j gives at least $(4 + \alpha - \sigma, 2 + \alpha)$ which dominates $(3 + \alpha + \sigma, 2 + \alpha)$.

Case 5. A is a top and-gate fed by 2-variables x_i and x_j.

Case 5.1. Variables x_i and x_j feed the same two gates.



Case 5.1.1



Case 5.1.2

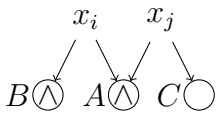
Case 5.1.1. B is an xor-gate. An affine substitution $x_i \leftarrow x_j \oplus c$ eliminates the dependence on x_j in the branch where A becomes constant, so we get at least $(\alpha, 2\alpha)$.

Case 5.1.2. B is an and-gate. Similarly to the Case 3 of the proof of Lemma 11 we get $(\alpha, 2\alpha)$.

Case 5.2. Variables x_i and x_j feed three gates: A, B, and C.

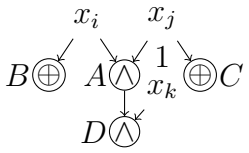
Note that in the following cases eliminating gates B and C we can not kill a 1-variable, otherwise we would be either in the Case 1 of in the Case 3.

Case 5.2.1. x_i and x_j feed two and-gates. W.l.o.g., B is an and-gate.

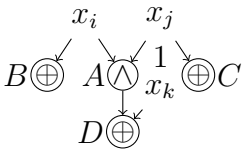


Assigning a constant to x_i we trivialize gates A and B in one of the branches, so we eliminate either four gates in one branch and two in the other one, or three gates in both branches. This gives either $(4+\alpha-2\sigma, 2+\alpha)$ or $(3+\alpha-\sigma, 3+\alpha-\sigma)$, which dominate $(3+\alpha+\sigma, 2+\alpha)$ and $(3+\alpha-2\sigma, 3+\alpha-2\sigma)$ respectively.

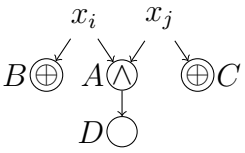
Case 5.2.2. Both B and C are xor-gates.



Case 5.2.2.1



Case 5.2.2.2



Case 5.2.2.3

Case 5.2.2.1. A is a 1-gate and its only successor D is an and-gate fed by the 1-variable x_k. Assigning constants to x_i and x_j we eliminate also the dependence on x_k in one of the branches. We get at least $(2\alpha, 2\alpha, 2\alpha, 3\alpha)$.

Case 5.2.2.2. A is a 1-gate and its only successor D is an xor-gate

fed by the 1-variable x_k. Let gate A computes function $(x_i \oplus a_A)(x_j \oplus b_A) \oplus c_A$. An affine substitution $x_k \leftarrow (x_i \oplus a_A)(x_j \oplus b_A) \oplus c$ makes D constant and eliminates at least three gates in each branch; in addition x_i and x_j become 1-variables. So, we get at least $(3+\alpha+\sigma, 3+\alpha+\sigma)$ which dominates $(3+\alpha+\sigma, 2+\alpha)$. Note that x_k only feeds gate D which is now constant so we do not need to replace x_k by a subcircuit computing $(x_i \oplus a_A)(x_j \oplus b_A) \oplus c$.

Case 5.2.2.3. A is either a 2^+-gate or a 1-gate with the only successor D which is not fed by a 1-variable. Assigning x_i a constant we eliminate three gates in one branch and two in the other one, in addition x_j becomes a 1-variable in the branch where A becomes constant. Gate D does not introduce new 1-variables, otherwise we would be in one of the previous two cases. Thus, we get $(3+\alpha+\sigma, 2+\alpha)$.

□

Corollary 7. 1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over B_2 of size at most $(2.6-\epsilon)n$ can be solved in time $(2-\delta)^n$.

2. Let $f \in B_n$ be an $(n, r(n) = n - o(n))$-quadratic disperser. Then $C_{B_2}(f) \geq 3n - o(n)$.

3. Let $f \in B_n$ be an $\left(n, r(n) = n - o(n), \epsilon(n) = 2^{-\omega(\log n)}\right)$-quadratic extractor. Then $C_{B_2}(f, \delta) \geq 2.6n - t$, where $\delta = 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(t-7.8(n-r(n)))^2}{121.68r(n)}\right)$.

This, in particular, implies that $\mathrm{Cor}(f, C)$ is negligible for any circuit C of size $2.6n - g(n)$ for some $g(n) = o(n)$.

Proof. 1. Let $\sigma = 1/5$. First note that for large enough α, we have

$$\begin{aligned} \tau(\alpha, 2\alpha) &< \tau(2\alpha, 2\alpha, 2\alpha, 3\alpha) &< \tau(2.6+\alpha, 2.6+\alpha) = 2^{\frac{1}{2.6+\alpha}} \\ &< \tau(3.5+\alpha, 3+\alpha) &< \tau(3.2+\alpha, 2+\alpha). \end{aligned}$$

Let $\gamma(\alpha) = \tau(3.2+\alpha, 2+\alpha) - 2^{\frac{1}{2.6+\alpha}}$. By Lemma 3, $\gamma(\alpha) = O(1/\alpha^3)$ holds. The running time of the algorithm is at most

$$\begin{aligned} (\tau(3.2+\alpha, 2+\alpha))^{s+\alpha n} &\leq \left(2^{\frac{1}{2.6+\alpha}}(1+\gamma(\alpha))\right)^{s+\alpha n} &\leq 2^{\frac{s+\alpha n}{2.6+\alpha}} 2^{(s+\alpha n)\gamma(\alpha)\log_2 e} \\ &\leq 2^{\frac{(2.6-\epsilon)n+\alpha n}{2.6+\alpha} + O(n/\alpha^2)} &\leq (2-\delta)^n \end{aligned}$$

for some $\delta > 0$ if we set $\alpha = c/\epsilon$ for large enough $c > 0$.

2. Lemma 13 guarantees that for $\alpha = 6$, $\sigma = 0$ one can always make an affine substitution reducing $s + 6i$ by at least 9. The function f is resistant to $r(n)$ such substitutions. Hence for a circuit C computing f, $s(C) + 6n \geq 9r(n)$.

3. Let us consider a circuit C of size at most $2.6n - t$, that is, $\mu(C) \leq (2.6n - t) + \alpha n$. Now we fix $\alpha = 5.2$, $\sigma = 0.2$, then $\beta_a = \min\{7.8, 11.7, 7.8, 7.8\} = 7.8, \beta_m = \min\{5.2, 5.2, 7.8, 7.2\} = 5.2$.

   We use the third item of Theorem 6 with $k = 2$, $r = r(n)$, $\epsilon = \epsilon(n)$,

$\mu = (2.6n - t + 5.2n)$, which gives us

$$\begin{aligned}\delta &= \epsilon(n) + \exp\left(\frac{-(7.8r(n)-(7.8n-t))^2}{2r(n)\cdot 2.6^2\cdot 3^2}\right) \\ &= \epsilon(n) + \exp\left(\frac{-(t-7.8(n-r(n)))^2}{121.68r(n)}\right).\end{aligned}$$

□

Remark 1. Note that it is an open problem to find an explicit construction of quadratic disperser or extractor over $\mathbb{F}_2$ with $r = n - o(n)$. It is shown in Section 5, that a disperser for a slightly more general definition of quadratic varieties would also imply a new worst case lower bound.

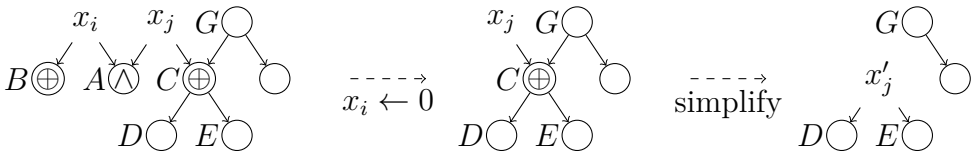
Remark 2. Note that the upper bound for #SAT can be improved using the following "forbidden trick", that is, a simplification rule that reduces the size of a circuit without changing the number of its satisfying assignments, but changes the function computed by the circuit.

In the proof of Lemma 13 set $\sigma = 0$ (that is, do not account for 1-variables). The set of splitting vectors then turn into

$$\{(\alpha, 2\alpha), (2\alpha, 2\alpha, 2\alpha, 3\alpha), (3+\alpha, 3+\alpha), (3+\alpha, 2+\alpha)\}.$$

By inspecting all the cases, we see that the splitting vector $(3+\alpha, 2+\alpha)$ only appears in the the Case 5.2.2. We can handle this case differently: split on x_i. When A is trivialized, x_j becomes a 1-variable feeding an xor-gate. It is not difficult to show that by replacing this gate with a new variable x'_j one gets a

circuit with the same number of satisfying assignments.



This additional trick gives us the following set of splitting vectors:

$$\{(\alpha, 2\alpha), (2\alpha, 2\alpha, 2\alpha, 3\alpha), (3+\alpha, 3+\alpha), (4+\alpha, 2+\alpha)\}.$$

These splitting numbers give an algorithm solving #SAT in $(2-\delta(\epsilon))^n$ for B_2-circuits of size at most $(3-\epsilon)n$ for $\epsilon > 0$.

7

Limitations of Gate Elimination

7.1 Overview

It is tempting to conjecture that the gate elimination method cannot eliminate many gates because it only changes the top part of a circuit. In general, this intuition fails for the following reason. Consider a function f of the highest circuit complexity $C(f) \geq 2^n/n$.[100] Every substitution (e.g., the substitution $x_1 = 0$) turns f into a function of $n-1$ variables. Since the circuit complexity of a function of $n-1$ variables cannot exceed $2^{n-1}/(n-1) + o(2^{n-1}/(n-1))$,[70] this substitution decreased the circuit complexity of b by almost a factor of two, i.e., it eliminated an exponential number of gates.

However, in this chapter we manage to make this intuition work for specially designed functions that compose gadgets satisfying certain rather general properties with arbitrary base functions. We show that certain formalizations of the gate elimination method cannot prove superlinear lower bounds. We prove that one cannot reduce the complexity of the designed functions by more than a constant using any constant number of substitutions of any type (that is, we allow to substitute variables by arbitrary functions). The complexity of a function may be counted as a complexity measure (i.e., a nonnegative function of a circuit) varying from the number of gates to any subadditive function. For recently popular measures that combine the number of gates with the number of inputs we prove a stronger result: One cannot prove lower bounds beyond cn for a certain specific constant c; this constant may depend on the number m of consecutive substitutions made in one step of the induction but does not depend on the substitutions themselves, in all modern proofs $m = 1$ or 2.

It was shown in Chapter 6 that the gate elimination method can also be used for proving average-case circuit lower bounds and upper bounds for Circuit #SAT. The limitation result of this chapter also applies to this line of research, implying that gate elimination cannot lead to strong improvements on the currently known results.

We summarize the known lower bound proofs in the table below (where the class $Q^n_{2,3}$ consists of functions that have at least three different subfunctions with respect to any two variables).

Bound	Class	Measure	Substitutions
$2n$[95]	$Q^n_{2,3}$	G	$x_i \leftarrow c$
$2.5n$[102]	symmetric	G	$x_i \leftarrow c$, $\{x_i \leftarrow f, x_j \leftarrow f \oplus 1\}$
$3n$[14]	artificial	G	arbitrary: $x_i \leftarrow f$
$3n$[29]	affine disp.	$G + I$	linear: $x_i \leftarrow \bigoplus_{j \in J} x_j \oplus c$
$3.01n$[37]	affine disp.	$G + \alpha I + \cdots$	linear: $x_i \leftarrow \bigoplus_{j \in J} x_j \oplus c$
$3.1n$[41]	quadratic disp.	$G + \alpha I$	quadratic: $x_i \leftarrow f$, $\deg(f) \leq 2$

It is interesting to note that there is a trivial limitation for the first three proofs in the table above: the corresponding classes of functions contain functions of linear circuit complexity. The class $Q^n_{2,3}$ contains the function THR^n_2 (that outputs 1 if and only if the sum of n input bits is at least 2) of circuit size $2n + o(n)$. The class of symmetric functions used by Stockmeyer contains the function MOD^n_4 whose circuit size is at most $2.5n + \Theta(1)$. The circuit size of Blum's function is upper bounded by $6n + o(n)$. At the same time it is not known whether there are affine dispersers of sublinear dimension that can be computed by linear size circuits.

7.2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. A *substitution* ρ of a set of variables $R \subseteq X$ is a set of $|R|$ restrictions of the form

$$r_i = f_i(x_1, \ldots, x_n),$$

one restriction for each variable $r_i \in R$, where f_i depends only on variables from $X \setminus R$. The degree of a substitution is the maximum degree of f_i's represented as Boolean polynomials. The size of a substitution is $|R|$. Substitutions of size m are called m-substitutions.

Given an m-substitution ρ and a function f, one can naturally define a new function $f|_\rho$ that has m fewer arguments than f.

A function f *depends* on a variable x if there is a substitution ρ of constants to all other variables such that $f|_\rho(0) \neq f|_\rho(1)$.

A gate elimination argument uses a certain nonnegative complexity measure μ, a family of substitutions $\mathcal{S}$, a family of functions $\mathcal{F}$, a function $\texttt{gain}\colon \mathbb{N} \to \mathbb{R}$, and a certain predicate $\texttt{stop}$, and includes proofs of the following statements:

1. (Measure usefulness.) If $\mu(f)$ is large, then $G(f)$ is large.

2. (Invariance.) For every $f \in \mathcal{F}$ and $\rho \in \mathcal{S}$, either $f|_\rho \in \mathcal{F}$ or $\texttt{stop}(f|_\rho)$.

3. (Induction step.) For every $f \in \mathcal{F}$ with $I(f) = n$, there is a substitution $\rho \in \mathcal{S}$ such that $\mu(f|_\rho) \leq \mu(f) - \texttt{gain}(n)$. (In known proofs, $\texttt{gain}(n)$ is constant.)

The family must contain functions f such that $\texttt{stop}(f|_{\rho_1,\ldots,\rho_s})$ is not reached for sufficiently many substitutions from $\mathcal{S}$ (for example, for $s = 0.999 \cdot I(f)$ substitutions).

In what follows, we prove that every gate elimination argument fails to prove a strong lower bound, for many functions of (virtually) arbitrarily large complexity.

7.3 Introductory example

We start by providing an elementary construction of functions that are resistant with respect to any constant number of arbitrary substitutions, i.e., such substitutions eliminate only a constant number of gates. In the next sections, we generalize this construction to capture other complexity measures.

Consider a function $f \in B_n$ and let $f \diamond \mathrm{MAJ}_3 \in B_{3n}$ be a function resulting from f by replacing each of its input variables x_i by the majority function of three fresh variables x_{i1}, x_{i2}, x_{i3} (see Figure 7.1):

$$(f \diamond \mathrm{MAJ}_3)(x_{11}, x_{12}, \ldots, x_{n3}) = f(\mathrm{MAJ}_3(x_{11}, x_{12}, x_{13}), \ldots, \mathrm{MAJ}_3(x_{n1}, x_{n2}, x_{n3}))\,.$$

Consider a circuit C of the smallest size computing $f \diamond \mathrm{MAJ}_3$. We claim that no substitution $x_{ij} \leftarrow \rho$, where ρ is any function of all the remaining variables, can remove from C more than 5 gates: $G(C) - G(C|_{x_{ij} \leftarrow \rho}) \leq 5$. We are going to prove this by showing that one can attach a gadget of size 5 to the circuit $C|_{x_{ij} \leftarrow \rho}$ and obtain a circuit that computes f. This is explained in Fig. 7.2. Formally, assume, without loss of generality, that the substituted variable is x_{11}. We then take a circuit C' computing $f|_{x_{11} \leftarrow \rho}$ and use the value of a gadget computing $\mathrm{MAJ}_3(x_{11}, x_{12}, x_{13})$ instead of x_{12} and x_{13}. This way we suppress the effect of the substitution $x_{11} \leftarrow \rho$, and the resulting circuit C'' computes the initial function $f \diamond \mathrm{MAJ}_3$. Since the majority of three bits can be computed in five gates, we get:

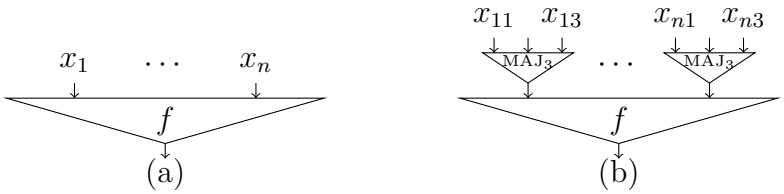


Figure 7.1: (a) A circuit for f. (b) A circuit for $f \diamond \mathrm{MAJ}_3$.

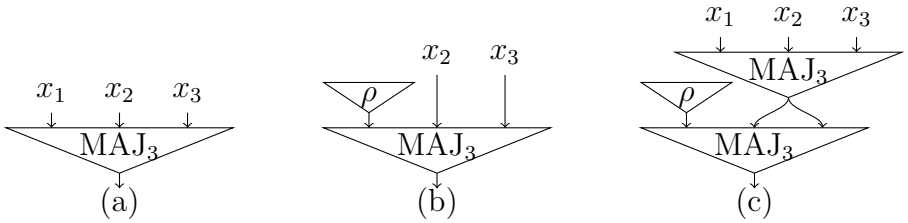


Figure 7.2: (a) A circuit computing the majority of three bits x_1, x_2, x_3. (b) A circuit resulting from substitution $x_1 \leftarrow \rho$. (c) By adding another gadget to a circuit with x_1 substituted, we force it to compute the majority of x_1, x_2, x_3.

$$G(C) \leq G(C'') \leq G(C|_{x_{11} \leftarrow \rho}) + 5\,.$$

This trick can be extended from 1-substitution to m-substitutions in a natural way. For this, we use gadgets computing the majority of $2m+1$ bits instead of just three bits. We can then suppress the effect of substituting any m variables by feeding the values to $m+1$ of the remaining variables. Taking into account the fact that the majority of $2m+1$ bits can be computed by a circuit of size $4.5(2m+1)$[30], we get the following result.

Lemma 14. For any $h \in B_n$ and any $m > 0$, the function $f = h \diamond \mathrm{MAJ}_{2m+1} \in B_{n(2m+1)}$ satisfies the following two properties:

- Circuit complexity of f is close to that of h: $G(h) \leq G(f) \leq G(h) +$

$4.5(2m+1)n$,

- For any m-substitution ρ, $G(f) - G(f|_\rho) \leq 4.5(2m+1)m$.

Remark 1. Note that from the Circuit Hierarchy Theorem (see, e.g.,[55]), one can find h of virtually any circuit complexity from n to $2^n/n$.

7.4 Subadditive measures

In this section we generalize the result of Lemma 14 to *arbitrary* subadditive measures. A function $\mu\colon B_n \to \mathbf{R}$ is called a *subadditive complexity measure*, if for all functions f and g, $\mu(h) \leq \mu(f) + \mu(g)$, where $h(\bar{x}, \bar{y}) = f(g(\bar{x}), \ldots, g(\bar{x}), \bar{y})$. That is, if h can be computed by application some function g to some of the the inputs, and then evaluating f, then the measure of h must not exceed the sum of measures of f and g. Clearly, the measures $\mu(f) = G(f)$ and $\mu_\alpha(f) = G(f) + \alpha \cdot I(f)$ are subadditive, and so are many other natural measures.

Let $f \in B_n$ and $g \in B_k$. Then by $h = f \diamond g \in B_{nk}$ we denote the function resulting from f by replacing each of its input variables by h applied to k fresh variables.

Our main construction is such a composition of a function f (typically, of large circuit complexity) and a gadget g that is chosen to satisfy certain combinatorial properties. Note that since we show a limitation of the proof method rather than a proof of a lower bound, we do not necessarily need to present explicit functions.

In this section we use gadgets that satisfy the following requirement: For every set of variables Y of size m, we can force the value of the gadget to be 0 and 1 by assigning constants only to the remaining variables.

Definition 9 (weakly m-stable function)**.** A function $g(X)$ is weakly m-stable if, for every $Y \subseteq X$ of size $|Y| \leq m$, there exist two assignments $\tau_0, \tau_1 \colon X \setminus Y \to \{0,1\}$ to the remaining variables, such that $g|_{\tau_0}(Y) \equiv 0$ and $g|_{\tau_1}(Y) \equiv 1$. That is, after the assignment τ_0 (τ_1), the function does not depend on the remaining variables Y.

It is easy to see that MAJ_{2m+1} is a weakly m-stable function. In Lemma 15 we show that almost all Boolean functions satisfy an even stronger requirement of stability.

Theorem 7. Let μ be a subadditive measure, $f \in B_n$ be any function, $g \in B_k$ be a weakly m-stable function, and $h = f \diamond g \in B_{nk}$. Then for every m-substitution ρ, $\mu(h) - \mu(h|_\rho) \leq m \cdot \mu(g)$.

Proof. Similarly to Lemma 14, we use a circuit H for the function $h|_\rho$ to construct a circuit C for h. Let

$$h(x_{11}, x_{12}, \ldots, x_{nk}) = f(g(x_{11}, \ldots, x_{1k}), \ldots, g(x_{n1}, \ldots, x_{nk})).$$

Let us focus on the variables $x_{11}, \ldots, x_{1k}$. Assume, without loss of generality, that the variables $x_{11}, \ldots, x_{1r}$ are substituted by ρ. Since ρ is an m-substitution, $r \leq m$. From the definition of weakly m-stable function, there exist substitutions τ_0 and τ_1 to the variables $x_{1r+1}, \ldots, x_{1k}$, such that $g|_{\rho\tau_0} = 0$ and $g|_{\rho\tau_1} = 1$.

We take the circuit H and add a circuit computing $g(x_{11}, \ldots, x_{1k})$. Now, for every variable $x \in \{x_{1r+1}, \ldots, x_{1k}\}$ in the circuit H, we wire $g(x_{11}, \ldots, x_{1k}) \oplus \tau_0(x)$ instead of x if $\tau_0(x) \neq \tau_1(x)$, and wire $\tau_0(x)$ otherwise. That is, we set $x_{1r+1}, \ldots, x_{1k}$ in such a way that $g|_\rho(x_{1r+1}, \ldots, x_{1k}) = g(x_{11}, \ldots, x_{1k})$. Thus, we added one instance of a circuit computing the gadget g and "repaired" $g(x_{11}, \ldots, x_{1k})$.

Now we repeat this procedure for each of the n inner functions g that have at least one variable substituted by ρ. Since ρ is an m-substitution, there are at most m gadgets we need to repair. Thus, we can compute h using the circuit H and m instances of a circuit computing g. From subadditivity of μ, $\mu(h) - \mu(h|_\rho) \leq m \cdot \mu(g)$. □

Corollary 8. Let $m = cn/2$, f be a Boolean function from $\{0,1\}^n$ to $\{0,1\}$, g be a weakly m-stable function, and $h = f \diamond g$ ($h : \{0,1\}^N \to \{0,1\}$ where $N \approx cn^2$). Then for every m-substitution ρ, $G(h) - G(h|_\rho) \leq O(N)$.

Using similar constructions and error correcting codes we can extend this corollary to larger substitutions.

Theorem 8. Let S be an error-correcting code with relative distance 2ϵ, codeword length N and message length n. Let $D(x)$ be a Boolean circuit of size $d(n)$ decoding S correcting ϵn errors, and $E(x)$ be a Boolean circuit of size $e(N)$ encoding S. Let f be a Boolean function from $\{0,1\}^n$ to $\{0,1\}$ and let $h = f \circ D$ ($h : \{0,1\}^N \to \{0,1\}$) be a composition of f and D. Then

1. $G(h) \geq G(f) - e(n)$,

2. for every $\epsilon \cdot n$-substitution ρ, $G(h) - G(h|_\rho) \leq e(n) + d(N)$.

Proof. Let us prove that $G(h) \geq G(f) - e(n)$. Let C be a circuit computing h. Let us consider the composition of two circuits $C'(x) = C(E(x))$. Note that C' computes f and the size of C' equals $G(h) + e(n)$.

Now let us prove that for every $\epsilon \cdot n$-substitution ρ, $G(h) - G(h|_\rho) \leq e(n) + d(N)$. Let C be a circuit computing $h|_\rho$. Let us consider the circuit $C''(x) = C(E(D(x)))$ (to abuse the notation, we assume that C just ignores the substituted inputs). Note that C'' computes h and the size of C'' equals $G(h|_\rho) + e(n) + d(N)$. □

Corollary 9. For any $\epsilon > 0$, there is a function $g_n \in B_{N,n}$ (where N depends on n) such that for any Boolean function $f \in B_n$ and $h = f \circ g_n$ it holds that

1. $G(h) \geq G(f) - O(n)$,

2. for every $(\frac{1}{2} - \epsilon) \cdot N$-substitution ρ, $G(h) - G(h|_\rho) \leq O(N \log(N))$.

Proof. Results of[43] and classical transformation from Turing machines to circuits shows that for any $\epsilon > 0$ there is a error-correcting code $C : \{0,1\}^n \to \{0,1\}^N$ ($N = O(n)$) with distance $(1 - \epsilon) \cdot N$, and circuits of size $O(n)$ and $O(n \log(n))$ for encoding and decoding respectively.

The corollary then follows from Theorem 8. □

Remark 2. To complement the result from Corollary 9, we note that any function $h \colon \{0,1\}^N \to \{0,1\}$ can be trivialized by $N/2$ substitutions.

Corollary 10. There exists a function $f \in B_n$ such that any decision tree of f has size at least $2^{\Omega(n)}$ even if branchings $x \leftarrow \rho$ and $x \leftarrow \rho \oplus 1$ (where ρ is an arbitrary function) are allowed.

Proof. Use any function of exponential complexity and apply Corollary 9. □

7.5 Measures that count inputs

The number of gates is not the only circuit complexity measure that is used in gate elimination proofs. In some bottleneck cases, it is not possible to find a substitution killing many gates, but it is still possible to make a substitution that reduces some other complexity parameter of a circuit significantly. One such parameter is the number of inputs of a circuit. In[29] it is used as follows. Assume that two variables x and y feed an $\wedge$-gate. If one of them (say, x) has out-degree at least 2, one easily eliminates at least three gates: assign $x \leftarrow 0$, this kills all successors of x (at least two gates) and also makes the $\wedge$-gate constant, so its successors are also eliminated (at least one gate). If, on the other hand, both x and y have out-degree 1, it is not clear how to eliminate more than two gates by assigning x or y. One notes, however, that the substitution $x \leftarrow 0$ eliminates not only two gates, but also two inputs: x is assigned, while y is just not needed anymore as the only gate that is fed by y turns into a constant under $x \leftarrow 0$. If one deals with a function that is resistant w.r.t. any $n - k$ substitutions (and usually $k = o(n)$), then the situation like the one above (when by assigning one variable one makes a circuit independent of some other variable, too) can only appear k times. Indeed, if k such substitu-

tions can be made, then the circuit (and hence the function) trivializes after $k + (n - 2k) = n - k$ substitutions (contradicting the fact that it is stable to any $n - k$ substitutions). Usually we have $k = o(n)$ which implies that this situation happens only $o(n)$ times. A convenient way of exploiting this fact is to incorporate the number of inputs into the circuit complexity measure. Namely,[29] uses the following measure: $\mu(C) = G(C) + I(C)$. Then, to prove a lower bound $G(C) \geq 3n - o(n)$ it is enough to prove that $\mu(C) \geq 4n - o(n)$. For this, in turn, one shows that it is always possible to find a substitution that reduces μ by at least 4. For the two cases discussed above it is easy: in the former case, we remove three gates and one input (hence, μ is reduced by 4), in the latter one, we remove two gates and two inputs (μ is reduced by 4 again). In[42,41] a more general measure is used: $\mu_\alpha(C) = G(C) + \alpha \cdot I(C)$, where $\alpha > 0$ is a constant. A typical m-substitution reduces μ_α by $k + \alpha m$ where k is the number of gates eliminated. If, however, a substitution removes more than m inputs, then μ_α is reduced by at least $\alpha(m + 1)$. By choosing a large enough value for α, one ensures that $\alpha(m + 1) \geq k + \alpha m$.

For example, in Lemma 14 we show that there are circuits where no substitution can eliminate too many gates. But this claim does not exclude the following possibility: Assume that for some circuits we can eliminate $\log n$ gates, and for the remaining circuits we cannot eliminate even 2 gates, but we can eliminate 2 inputs. Then by setting $\alpha \approx \log n$ and considering the measure $\mu_\alpha(C) = G(C) + \alpha \cdot I(C)$ one would prove a superlinear lower bound.

In this section, we construct gadgets against such measures. Namely, we con-

struct a function f such that any m-substitution reduces the number of gates by a constant c_m and reduces the number of inputs by m. This prevents anyone from proving a better than $c_m n$ bound using these measures.

Definition 10 (m-stable function)**.** A function $g(X)$ is m-stable if, for every $Y \subseteq X$ of size $|Y| \leq m+1$ and every $y \in Y$, there exists an assignment $\tau\colon X \setminus Y \to \{0,1\}$ to the remaining variables such that $g|_\tau(Y) \equiv y$ or $g|_\tau(Y) \equiv \neg y$. That is, after the assignment τ, the function depends only on the variable y.

It is now easy to see that every m-stable function is weakly m-stable.

Theorem 9. Let f be a Boolean function, g be an m-stable function, and $h = f \diamond g$. Then for every m-substitution ρ, $\mu_\alpha(h) - \mu_\alpha(h|_\rho) \leq m \cdot (G(g) + \alpha)$.

Proof. Since g is m-stable, Theorem 7 implies that $G(h) - G(h|_\rho) \leq m \cdot G(g)$. It remains to show that $I(h) - I(h|_\rho) = m$. Thus, it suffices to prove that if f depends on x_{ij} and ρ does not substitute $x_{i,j}$, then $h|_\rho$ depends on $x_{i,j}$. Let

$$h(x_{11}, x_{12}, \ldots, x_{nk}) = f(g(x_{11}, \ldots, x_{1k}), \ldots, g(x_{n1}, \ldots, x_{nk})).$$

Without loss of generality let $i = 1$. Let R be the set of variables x_{st} for $s > 1$ substituted by ρ. There exists a substitution η to the variables $\{x_{21}, \ldots, x_{2k}, \ldots, x_{n1}, \ldots, x_{nk}\} \setminus R$ such that $h|_\eta(x_{11}, \ldots, x_{1k})$ does not depend on the variables in R and is not a constant: by the definition of m-stability we can force the instances of the gadget g except for the first one to produce any desired assignment for the inputs of f (all but the first one).

Let us consider the variables $x_{11}, \ldots, x_{1k}$. Assume, without loss of generality, that the variables $x_{11}, \ldots, x_{1r}$ are substituted by ρ. Since ρ is an m-substitution, $r \leq m$. Now we want to show that for every $j > r$, $h|_\rho$ depends on x_{1j}. From the definition of an m-stable function, there exists a substitution τ to $\{x_{1,r+1}, \ldots, x_{1k}\} \setminus \{x_{1j}\}$ such that $g|_{\rho\tau}(x_{1j})$ is not constant ($g|_{\rho\tau} = x_{1j}$ or $g|_{\rho\tau} = \neg x_{1j}$). Now, we compose the substitutions η and τ, which gives us that $h|_{\rho\tau\eta}(x_{1j})$ is not constant. This implies that the function $h|_\rho$ depends on the variable x_{1j}. □

Now we show that for a fixed m, almost all Boolean functions are m-stable.

Lemma 15. For $m \geq 1$ and $k = \Omega(2^m)$, a random $f \in B_k$ is m-stable almost surely.

Proof. Let X denote the set of k input variables. Let us fix a set Y, $|Y| \leq m+1$, and a variable $y \in Y$. Now let us count the number of functions that do not satisfy the definition of m-stable function for this fixed choice of Y and y. Thus, for each assignment to the variables from $X \setminus Y$, the function must not be y nor $\neg y$. There are 2^{k-m-1} assignments to the variables $X \setminus Y$, and at most $(2^{2^{m+1}} - 2)$ functions of $(m+1)$ variables that are not y nor $\neg y$. Thus, there are at most $(2^{2^{m+1}} - 2)^{2^{k-m-1}}$ functions that do not satisfy the definition of m-stable function for this fixed choice of Y and y. Now, since there are $\binom{k}{m+1} \cdot (m+1)$ ways to choose Y and y, the union bound implies that a random function is not m-stable with probability at most

$$\frac{\binom{k}{m+1}(m+1)(2^{2^{m+1}}-2)^{2^{k-m-1}}}{2^{2^k}} \leq k^{m+2} \cdot \left(\frac{2^{2^{m+1}}-2}{2^{2^{m+1}}}\right)^{2^{k-m-1}} \leq$$
$$\exp\left((m+2)\ln k - 2^{k-m-2^{m+1}}\right) = o(1)$$

for $k = \Omega(2^m)$. □

Lemma 15, together with Theorem 9, provides a class of functions such that any m-substitution decreases the measure μ_α by at most a fixed constant (which may depend on m but not on α).

Corollary 11. For any $m > 0$, there exist $k > 0$ and $g \in B_k$ such that for any f of n inputs, the function $h = f \diamond g \in B_{nk}$ satisfies:

- Circuit complexity of h is close to that of f: $G(f) \leq G(h) \leq G(f) + G(g) \cdot n$,
- For any m-substitution ρ and real $\alpha > 0$, $\mu_\alpha(h) - \mu_\alpha(h|_\rho) \leq G(g) \cdot m + \alpha m$.

Thus, gate elimination with m-substitutions and μ_α measures can prove only $O(n)$ lower bounds.

Although Lemma 15 proves the existence of m-stable functions, their circuit complexities may be large (though constant). To optimize these constants, one can use explicit constructions of m-stable functions.

Lemma 16. For any m, there exists an m-stable function of circuit complexity at most $O(m^2 \log m)$.

Proof. Let n be a power of two, and let $C\colon \{1,\dots,n\} \to \{0,1\}^n$ be the Walsh–Hadamard error correcting code (a code with distance $\frac{n}{2}$, see, e.g.,[8] Section 19.2.2). We define a function $g_C\colon \{0,1\}^n \to \{0,1\}$ in the following way. Given an input x, we first find the nearest codeword $C(i)$ to x (any of them in the case of a tie), and then output the ith bit of the input: $g_C(x) = x_i$.

It is easy to see that g_C can be computed in randomized linear time $O(n)$, thus, it can be computed by a circuit of size $O(n^2 \log n)$ (see, e.g.,[2]).

Let us show that g_C is $\left(\frac{n}{4} - 2\right)$-stable. To this end we show that for any set $Y \subseteq \{x_1,\dots,x_n\}, |Y| \leq \left(\frac{n}{4} - 1\right)$, for any $y \in Y$, there exists an assignment to the remaining variables that forces g_C to compute y. Without loss of generality, assume that $Y = \{x_1,\dots,x_{n/4-1}\}$ and that $y = x_1$. Let us fix the last $3n/4 + 1$ bits to be equal to the corresponding bits of $C(1)$. Namely, we set $(x_{n/4},\dots,x_n) = (C(1)_{n/4},\dots,C(1)_n)$. After these substitutions, any input x has distance less than $n/4$ to the codeword $C(1)$, thus $C(1)$ is the nearest codeword. This implies that $g_C(x)$ always outputs $y = x_1$. □

Corollary 12. For any $m > 0$, there exists a function g of $k = O(m)$ inputs such that for any function h of n inputs, the function $f = h \diamond g$ of nk inputs satisfies:

- Circuit complexity of f is close to that of h: $G(h) \leq G(f) \leq G(h) + O(m^2 n \log m)$,
- For any m-substitution ρ and real $\alpha > 0$, $\mu_\alpha(f) - \mu_\alpha(f|_\rho) \leq O(m^3 \log m) + \alpha m$.

A computer-assisted search gives a 1-stable function of 5 inputs that can be computed with 11 gates.

Lemma 17. There exists a 1-stable function $g_1^{\mathrm{st}}\colon \{0,1\}^5 \to \{0,1\}$ of circuit complexity at most 11.

Proof. The truth table of the function g_1^{st} is shown below.

x_0	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
x_1	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
x_2	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
x_3	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
x_4	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
$g_1^{\mathrm{st}}(x_0, x_1, x_2, x_3, x_4)$	0 0 0 0 0 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 0

It can be checked that for any $i, j \in \{0,1,2,3,4\}$, where $i \neq j$, there exist $c_1, c_2, c_3 \in \{0,1\}$ such that when the three remaining variables are assigned the values c_1, c_2, c_3, the function g_1^{st} turns into x_i. For example, under the substitution $\{x_0 \leftarrow 0, x_2 \leftarrow 0, x_4 \leftarrow 1\}$ the function g_1^{st} is equal to x_1. □

This lemma implies that the basic gate elimination argument is unable to prove a lower bound of $11n$ using the measure μ_α and 1-substitutions. (Note that almost all known proofs use either 1- or 2-substitutions.)

Corollary 13. For any function h of n inputs, assume function $f = h \diamond g_1^{\mathrm{st}}$ (it has $5n$ inputs). Then

1. The complexity of f is close to that of h: $G(h) \leq G(f) \leq G(h) + 11n$;

2. For any 1-substitution ρ and real $\alpha > 0$, $\mu_\alpha(f) - \mu_\alpha(f|_\rho) \leq 11 + \alpha$.

8

Conclusions

In this work, we have obtained new results using gate elimination and also showed limitations of this method. A natural further direction is to develop new methods for proving circuit lower bounds against Boolean circuits of unbounded depth. We summarize several specific open problems below.

- One of the few examples of lower bounds against circuits of unbounded depth which does not use gate elimination is the work of Chashkin[18]. He proved a lower bound of $2n - o(n)$ on the complexity of the parity-check matrix of Hamming codes. Another classical example of a lower bound which does not use gate elimination is a lower bound of Blum and

Seysen[15] who showed that an optimal circuit computing AND and OR must have two separate trees computing outputs (which also gives a lower bound of $2n - 2$). Melanich[72] proved a similar property and a lower bound of $2n - o(n)$ for a function whose outputs compute products of specific subsets of inputs. Can we extend these techniques to prove new stronger lower bounds?

- A natural question left open by this work is to find an explicit construction of a quadratic disperser. Such a construction would imply new circuit lower bounds (see Chapter 5 and Section 6.5.2). Another big open problem is to find explicit construction of dispersers for polynomial varieties of *higher* degrees. By Valiant's reductions[108], any $O(\log n)$-depth circuit for such a disperser must have superlinear size.

- Although the limitation result from Chapter 7 covers almost all currently used techniques, it is not fully general. It would be great to extend it by showing that the limitation holds for more general classes of circuit measures, and for all large enough classes of Boolean functions.

References

[1] Aaronson, S. & Wigderson, A. (2009). Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1), 2.

[2] Adleman, L. (1978). Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science* (pp. 75–83).: IEEE.

[3] Alon, N. & Spencer, J. H. (2004). *The probabilistic method*. John Wiley & Sons.

[4] Amano, K. & Saito, A. (2015a). A nonuniform circuit class with multi-layer of threshold gates having super quasi polynomial size lower bounds against NEXP. In *Proceedings of the 9th International Conference on Language and Automata Theory and Applications (LATA)* (pp. 461–472).

[5] Amano, K. & Saito, A. (2015b). A satisfiability algorithm for some class of dense depth two threshold circuits. *IEICE Transactions*, 98-D(1), 108–118.

[6] Amano, K. & Tarui, J. (2011). A well-mixed function with circuit complexity $5n$: Tightness of the Lachish-Raz-type bounds. *Theor. Comput. Sci.*, 412(18), 1646–1651.

[7] Andreev, A. E. (1987). 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.

[8] Arora, S. & Barak, B. (2009). *Computational complexity: a modern approach*. Cambridge.

[9] Baker, T., Gill, J., & Solovay, R. (1975). Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM Journal on computing*, 4(4), 431–442.

[10] Beame, P., Impagliazzo, R., & Srinivasan, S. (2012). Approximating AC^0 by small height decision trees and a deterministic algorithm for $\#AC^0$ SAT. In *Proceedings of the 27th Conference on Computational Complexity (CCC)* (pp. 117–125).

[11] Ben-Sasson, E. & Gabizon, A. (2013). Extractors for polynomial sources over fields of constant order and small characteristic. *Theory of Computing*, 9, 665–683.

[12] Ben-Sasson, E. & Kopparty, S. (2012). Affine dispersers from subspace polynomials. *SIAM J. Comput.*, 41(4), 880–914.

[13] Ben-Sasson, E. & Viola, E. (2014). Short PCPs with projection queries. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I* (pp. 163–173).

[14] Blum, N. (1984). A Boolean function requiring $3n$ network size. *Theor. Comput. Sci.*, 28, 337–345.

[15] Blum, N. & Seysen, M. (1984). Characterization of all optimal networks for a simultaneous computation of AND and NOR. *Acta informatica*, 21(2), 171–181.

[16] Buhrman, H., Fortnow, L., & Thierauf, T. (1998). Nonrelativizing separations. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC)* (pp. 8–12).

[17] Chakaravarthy, V. T. & Roy, S. (2011). Arthur and Merlin as oracles. *Computational Complexity*, 20(3), 505–558.

[18] Chashkin, A. V. (1994). On the complexity of Boolean matrices, graphs and their corresponding Boolean functions. *Diskretnaya matematika*, 6(2), 43–73.

[19] Chen, R. (2015). Satisfiability algorithms and lower bounds for Boolean formulas over finite bases. In *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II* (pp. 223–234).

[20] Chen, R. & Kabanets, V. (2015). Correlation bounds and #SAT algorithms for small linear-size circuits. In *Proceedings of the 21st International Conference on Computing and Combinatorics (COCOON)* (pp. 211–222).

[21] Chen, R., Kabanets, V., Kolokolova, A., Shaltiel, R., & Zuckerman, D. (2015). Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2), 333–392.

[22] Chen, R., Kabanets, V., & Saurabh, N. (2014). An improved deterministic #SAT algorithm for small De Morgan formulas. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II* (pp. 165–176).

[23] Chen, R. & Santhanam, R. (2015). Improved algorithms for sparse MAX-SAT and MAX-k-CSP. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT)* (pp. 33–45).

[24] Chen, R., Santhanam, R., & Srinivasan, S. (2016). Average-case lower bounds and satisfiability algorithms for small threshold circuits. In *Proceedings of the 31th Conference on Computational Complexity (CCC)* (pp. 1:1–1:35).

[25] Chor, B., Goldreich, O., Håstad, J., Friedman, J., Rudich, S., & Smolensky, R. (1985). The bit extraction problem of t-resilient functions (prelim-

inary version). In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)* (pp. 396–407).

[26] Cohen, G. & Shinkar, I. (2016). The complexity of DNF of parities. In *Proceedings of the 7th Innovations in Theoretical Computer Science (ITCS) Conference* (pp. 47–58).

[27] Cohen, G. & Tal, A. (2015). Two structural results for low degree polynomials and applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA* (pp. 680–709).

[28] Demenkov, E., Kojevnikov, A., Kulikov, A. S., & Yaroslavtsev, G. (2010). New upper bounds on the Boolean circuit complexity of symmetric functions. *Information Processing Letters*, 110(7), 264–267.

[29] Demenkov, E. & Kulikov, A. S. (2011). 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* (pp. 256–265).: Springer.

[30] Demenkov, E. & Kulikov, A. S. (2012). Computing All MOD-Functions Simultaneously. *Computer Science – Theory and Applications*, (pp. 81–88).

[31] Demenkov, E., Kulikov, A. S., Melanich, O., & Mihajlin, I. (2015). New lower bounds on circuit size of multi-output functions. *Theory Comput. Syst.*, 56(4), 630–642.

[32] Dinur, I. & Meir, O. (2016). Toward the KRW composition conjecture: cubic formula lower bounds via communication complexity. In *31st Conference on Computational Complexity (CCC)* (pp.3).

[33] Dodis, Y. (2000). *Exposure-resilient cryptography*. PhD thesis, Massachusetts Institute of Technology.

[34] Dvir, Z. (2012). Extractors for varieties. *Computational complexity*, 21(4), 515–572.

[35] Dvir, Z., Gabizon, A., & Wigderson, A. (2009). Extractors and rank extractors for polynomial sources. *Computational Complexity*, 18(1), 1–58.

[36] Dymond, P. W. & Cook, S. A. (1989). Complexity theory of parallel time and hardware. *Inf. Comput.*, 80(3), 205–226.

[37] Find, M. G., Golovnev, A., Hirsch, E. A., & Kulikov, A. S. (2016). A better-than-3n lower bound for the circuit complexity of an explicit function. In *57th IEEE Symposium on Foundations of Computer Science (FOCS)* (pp. 89–98). IEEE.

[38] Fomin, F. V., Grandoni, F., & Kratsch, D. (2009). A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5).

[39] Fortnow, L. (1994). The role of relativization in complexity theory. *Bulletin of the EATCS*, (pp. 1–15).

[40] Golovnev, A., Hirsch, E. A., Knop, A., & Kulikov, A. S. (2016a). On the Limits of Gate Elimination. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 (pp. 46:1–46:13).

[41] Golovnev, A. & Kulikov, A. S. (2016). Weighted gate elimination: Boolean dispersers for quadratic varieties imply improved circuit lower bounds. In *Proceedings of the 7th Innovations in Theoretical Computer Science (ITCS) Conference* (pp. 405–411).

[42] Golovnev, A., Kulikov, A. S., Smal, A. V., & Tamaki, S. (2016b). Circuit size lower bounds and #SAT upper bounds through a general framework. In *Mathematical Foundations of Computer Science (MFCS)* (pp. 45:1–45:16).

[43] Guruswami, V. & Indyk, P. (2005). Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. Information Theory*, 51(10), 3393–3400.

[44] Håstad, J. (1986). Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing* (pp. 6–20).: ACM.

[45] Håstad, J. (1998). The shrinkage exponent of de Morgan formulas is 2. *SIAM J. Comput.*, 27(1), 48–64.

[46] Håstad, J. (2014). On the correlation of parity and small-depth circuits. *SIAM J. Comput.*, 43(5), 1699–1708.

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

[48] Hrubeš, P., Jukna, S., Kulikov, A., & Pudlak, P. (2010). On convex complexity measures. *Theoretical Computer Science*, 411(16), 1842–1854.

[49] Impagliazzo, R., Kabanets, V., & Volkovich, I. (2017). The power of natural properties as oracles. *Electronic Colloquium on Computational Complexity (ECCC)*, 24, 23.

[50] Impagliazzo, R., Matthews, W., & Paturi, R. (2012). A satisfiability algorithm for AC^0. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (pp. 961–972).

[51] Impagliazzo, R. & Nisan, N. (1993). The effect of random restrictions on formula size. *Random Struct. Algorithms*, 4(2), 121–134.

[52] Impagliazzo, R., Paturi, R., & Schneider, S. (2013). A satisfiability algorithm for sparse depth two threshold circuits. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (pp. 479–488).

[53] Iwama, K. & Morizumi, H. (2002). An Explicit Lower Bound of $5n - o(n)$ for Boolean Circuits. In *Mathematical Foundations of Computer Science (MFCS)*, volume 2420 of *Lecture Notes in Computer Science* (pp. 353–364).: Springer.

[54] Jahanjou, H., Miles, E., & Viola, E. (2015). Local reductions. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part I* (pp. 749–760).

[55] Jukna, S. (2012). *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media.

[56] Khrapchenko, V. M. (1971). 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.

[57] Kloss, B. M. & Malyshev, V. A. (1965). Estimates of the complexity of certain classes of functions. *Vestn.Moskov.Univ.Ser.1*, 4, 44–51. In Russian.

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

[59] Kojevnikov, A. & Kulikov, A. S. (2006). 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 (pp. 11–17).: ACM Press.

[60] Kojevnikov, A. & Kulikov, A. S. (2010). Circuit complexity and multiplicative complexity of Boolean functions. In *Proceedings of the 6th Conference on Computability in Europe (CiE)* (pp. 239–245).

[61] Komargodski, I. & Raz, R. (2013). Average-case lower bounds for formula size. In *Proceedings of the 45th Symposium on Theory of Computing (STOC)* (pp. 171–180).

[62] Komargodski, I., Raz, R., & Tal, A. (2013). 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* (pp. 588–597).: IEEE Computer Society.

[63] Kullmann, O. (1999). New methods for 3-SAT decision and worst-case analysis. *Theor. Comput. Sci.*, 223(1-2), 1–72.

[64] Kullmann, O. (2009). Fundaments of branching heuristics. In A. Biere, M. Heule, H. van Maaren, & T. Walsh (Eds.), *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications* (pp. 205–244). IOS Press.

[65] Lachish, O. & Raz, R. (2001). Explicit lower bound of $4.5n - o(n)$ for Boolean circuits. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC)* (pp. 399–408).

[66] Lamagna, E. A. & Savage, J. E. (1973). *On the logical complexity of symmetric switching functions in monotone and complete bases.* Technical report, Brown University.

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

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

[69] Lipton, R. J. (2010). *The $\mathcal{P} = \mathcal{NP}$ Question and Gödel's Lost Letter.* Springer Science & Business Media.

[70] Lupanov, O. (1958). A circuit synthesis method. *Izv. Vuzov, Radiofizika*, 1(1), 120–130.

[71] Maurey, B. (1979). Espaces de Banach: Construction de suites symetriques. *C.R. Acad. Sci. Paris Ser. A-B.*, 288, 679–681.

[72] Melanich, O. (2012). Personal communication.

[73] Nagao, A., Seto, K., & Teruyama, J. (2015). A moderately exponential time algorithm for k-IBDD satisfiability. In *Proceedings of the 14th International Symposium, on Algorithms and Data Structures (WADS)* (pp. 554–565).

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

[75] Newman, I. & Wigderson, A. (1995). Lower bounds on formula size of boolean functions using hypergraph entropy. *SIAM Journal on Discrete Mathematics*, 8(4), 536–542.

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

[77] Nigmatullin, R. G. (1985). Are lower bounds on the complexity lower bounds for universal circuits? In *Fundamentals of Computation Theory* (pp. 331–340).: Springer.

[78] Nigmatullin, R. G. (1990). *Complexity lower bounds and complexity of universal circuits*. Kazan University.

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

[80] Oliveira, I. C. (2013). Algorithms versus circuit lower bounds. *Electronic Colloquium on Computational Complexity (ECCC)*, TR13-117.

[81] Paterson, M. & Zwick, U. (1993). Shrinkage of de Morgan formulae under restriction. *Random Struct. Algorithms*, 4(2), 135–150.

[82] Paturi, R., Saks, M. E., & Zane, F. (2000). Exponential lower bounds for depth three boolean circuits. *Computational Complexity*, 9(1), 1–15.

[83] Paul, W. J. (1977). A 2.5n-lower bound on the combinational complexity of Boolean functions. *SIAM J. Comput.*, 6(3), 427–443.

[84] Rao, A. (2009). Extractors for low-weight affine sources. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)* (pp. 95–101).

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

[86] Razborov, A. A. (1990). Applications of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1), 81–93.

[87] Razborov, A. A. (1992). On submodular complexity measures. In *Poceedings of the London Mathematical Society Symposium on Boolean Function Complexity* (pp. 76–83). New York, NY, USA: Cambridge University Press.

[88] Razborov, A. A. & Rudich, S. (1997). Natural proofs. *Journal of Computer and System Sciences*, 55(1), 24–35.

[89] Riedel, M. D. & Bruck, J. (2012). Cyclic boolean circuits. *Discrete Applied Mathematics*, 160(13-14), 1877–1900.

[90] Rivest, R. L. (1977). The necessity of feedback in minimal monotone combinational circuits. *IEEE Trans. Computers*, 26(6), 606–607.

[91] Sakai, T., Seto, K., Tamaki, S., & Teruyama, J. (2016). Bounded depth circuits with weighted symmetric gates: Satisfiability, lower bounds and compression. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016* (pp. 82:1–82:16).

[92] Santhanam, R. (2010). Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (pp. 183–192).

[93] Santhanam, R. (2012). Ironic complicity: Satisfiability algorithms and circuit lower bounds. *Bulletin of the EATCS*, 106, 31–52.

[94] Savinov, S. (2014). Upper bounds for the Boolean circuit satisfiability problem. Master Thesis defended at St.Peterburg Academic University of Russian Academy of Sciences. In Russian.

[95] Schnorr, C. (1974). Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. *Computing*, 13(2), 155–171.

[96] Schnorr, C. (1976). The combinational complexity of equivalence. *Theor. Comput. Sci.*, 1(4), 289–295.

[97] Seto, K. & Tamaki, S. (2013). A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2), 245–274.

[98] Shaltiel, R. (2011a). Dispersers for affine sources with sub-polynomial entropy. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (pp. 247–256).

[99] Shaltiel, R. (2011b). 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* (pp. 21–41). Springer.

[100] Shannon, C. E. (1949). The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28, 59–98.

[101] Shoup, V. & Smolensky, R. (1991). 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* (pp. 378–383).: IEEE Computer Society.

[102] Stockmeyer, L. J. (1977). On the combinational complexity of certain symmetric Boolean functions. *Mathematical Systems Theory*, 10, 323–336.

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

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

[105] Tal, A. (2015). #SAT algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-114.

[106] Vadhan, S. & Williams, R. (2013). Personal communication.

[107] Valiant, L. G. (1976). Universal circuits (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing* (pp. 196–203).: ACM.

[108] Valiant, L. G. (1977). Graph-theoretic arguments in low-level complexity. In *MFCS 1977* (pp. 162–176).

[109] Wang, F. (2011). NEXP does not have non-uniform quasipolynomial-size ACC circuits of $o(\text{loglog}n)$ depth. In *Proceedings of the 8th Annual Conference on Theory and Applications of Models of Computation (TAMC)* (pp. 164–170).

[110] Wegener, I. (1987). *The complexity of Boolean functions*. Wiley-Teubner.

[111] Williams, R. (2013a). Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3), 1218–1244.

[112] Williams, R. (2013b). Natural proofs versus derandomization. In *Proceedings of the 45th ACM Symposium on Theory of Computing Conference (STOC)* (pp. 21–30).

[113] Williams, R. (2014a). Algorithms for circuits and circuits for algorithms. In *Proceedings of the 29th Annual IEEE Conference on Computational Complexity (CCC)* (pp. 248–261).

[114] Williams, R. (2014b). New algorithms and lower bounds for circuits with linear threshold gates. In *Proceedings of the 46th Symposium on Theory of Computing (STOC)* (pp. 194–202).

[115] Williams, R. (2014c). Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1), 2.

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

[117] Yehudayoff, A. (2011). Affine extractors over prime fields. *Combinatorica*, 31(2), 245–256.

[118] Zwick, U. (1991). 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.