

An Explicit $3.5n - o(n)$ Size Lower Bound against General Boolean Circuits

Om Swostik Mishra*

University of Oxford

Rahul Santhanam†

University of Oxford

Abstract

We give a $3.5n - o(n)$ lower bound on the size of Boolean circuits over the B_2 basis computing an explicit Boolean function $F \in \mathsf{P}$. This improves the best previous lower bound of $3.1n - o(n)$ [34].

The proof of the lower bound uses the gate elimination technique, and builds heavily on ideas in previous works [19, 34]. The main conceptual innovation is to consider substitutions on an underlying space of *independent parameters* specifying values of input variables, rather than on the input variables themselves. This enables us to handle certain bottleneck situations by *reparameterisation* rather than substitution. As a consequence, we are also able to obtain a simpler lower bound proof than in [20, 34].

1 Introduction

1.1 Background and Motivation

One of the main challenges in complexity theory is to prove strong lower bounds against natural computational models. The model of Boolean circuits [51] is of particular interest for its simplicity and generality. A circuit C for a Boolean function F on n bits is a network of logical gates that computes F from the base functions $x_1 \dots x_n$ representing the n input bits. The size of a circuit C is the number of gates in the network. We consider circuits where any gate has fan-in two; there is no restriction on the fan-out.

It is a standard fact [47] that any computational problem solvable in deterministic time t can be computed by Boolean circuits of size $O(t \log(t))$. Thus, lower bounds on Boolean circuit size translate to lower bounds on computation time. This fact underlies Sipser’s influential *circuit lower bound approach* [52] to the celebrated NP vs P problem¹: show that some problem in NP does not have polynomial-size circuits.

On the one hand, Sipser’s approach avoided the relativization barrier to complexity lower bounds [3], which indicated that simulation and diagonalization methods used in uniform complexity in the 1970s might not in themselves be sufficient to resolve NP vs P and related complexity lower bound questions. On the other hand, it opened the way to various forms of combinatorial and algebraic analysis, which resulted in several important lower bound results for restricted circuit models [1, 22, 59, 27, 42, 43, 54].

However, for unrestricted Boolean circuits, progress has been much slower, and the best explicit lower bounds known are of the form cn for some small constant c . Here, by an “explicit” lower bound, we mean a lower bound for a Boolean function in NP . It was shown already by Shannon [51] that most Boolean functions on n variables require circuit size $\Omega(2^n/n)$, but it is

*E-mail: om.mishra@mansfield.ox.ac.uk

†E-mail: rahul.santhanam@cs.ox.ac.uk

¹Thus far untouched by AI tools!

unknown even if there are such functions computable in NP or even in EXP. If we relax the explicitness requirement to computability in MA with small advice or S_2P , lower bounds of the form n^k are known for every fixed k [45, 11, 12]. If we were able to show that for each k there is a language $L_k \in NP$ requiring circuits of size n^k , it would follow that NEXP requires super-polynomial size circuits, resolving a longstanding conjecture in complexity theory.

In this paper, we are concerned with lower bounds for Boolean functions in NP. Since the best known lower bounds are linear with a small constant factor, they are sensitive to the choice of the gate set used in the circuit model. There are two widely-studied circuit models of general Boolean circuits: B_2 circuits, where any Boolean function on at most 2 inputs can be used as a gate, and U_2 circuits, where any Boolean function on at most 2 inputs can be used as a gate, except the XOR function and its complement. Note that any B_2 circuit of size s can be simulated by a U_2 circuit of size $3s$, as the XOR function and its complement have U_2 circuits of size 3.

The best known circuit size lower bound for explicit functions against B_2 circuits is $3.1n - o(n)$, proved by Li and Yang [34], building on [17, 19, 20]. The best explicit lower bound against U_2 circuits is $5n - o(n)$ [33, 29].

There are several motivations to attack the question of explicit lower bounds. First, new lower bounds would represent progress in Sipser's circuit lower bound approach [52, 53] to separating P from NP. Second, circuit lower bounds have implications for *derandomization*, i.e., non-trivial simulations of randomized algorithms by deterministic or non-deterministic algorithms. If we could show an explicit lower bound of n^k for each k , we would get a simulation of BPP (probabilistic polynomial time) in non-deterministic sub-exponential time (with a small amount of non-uniformity) [40, 28]. Currently, no better simulation than the trivial exponential-time brute-force simulation is known. Third, circuit size lower bounds for natural functions are interesting in their own right, as a measure of the amount of *hardware* required to compute a Boolean function - this has implications for functionalities that can be implemented feasibly on electronic chips. Fourth, and perhaps most significantly, improved lower bounds often reflect an improved structural and analytical understanding of circuits, and hence of computation in its more general form.

1.2 The Main Result

The main result of this paper is an improved lower bound for explicit functions against B_2 circuits. As in previous work showing explicit lower bounds in this setting, our lower bound is for a Boolean function in P.

Theorem 1.1. *There is a Boolean function in P which requires B_2 circuits of size $3.5n - O(\log(n))$.*

Continuing the tradition of previous work on this question [17, 19, 20, 34], we use the *gate elimination* technique, and our lower bound is for *affine dispersers*, for which explicit constructions with excellent parameters are known [5, 36, 13, 37].

Let F be a Boolean function (with certain useful properties) for which we wish to prove a circuit lower bound of the form $cn - o(n)$ for some integer c , and let C be a B_2 circuit computing F . In its simplest form, the gate elimination technique argues iteratively that we can set one of the input variables of C to a constant, or more generally to a function of the remaining variables, so that the induced Boolean function F' is not trivial (i.e., not a constant), and can be computed by a simplified Boolean circuit C' where c gates of C have been eliminated. If gate elimination can be done iteratively for $n - o(n)$ steps, a lower bound of $cn - o(n)$ immediately follows on the circuit size of C , as $c(n - o(n))$ gates have been eliminated and the resulting circuit still computes a non-trivial function.

More generally, lower bounds of the form $cn - o(n)$ for non-integral c are shown by doing an amortised analysis of the gains achieved by setting a variable. This is implemented using an appropriately defined potential function depending on key structural quantities of the circuit, such as the number of gates and the number of inputs². The improvements of [17, 19, 34] depend crucially on the analysis of an appropriate potential function. In this work, we generalise gate elimination further by setting *independent parameters* in an underlying parameter space in terms of which variable assignments are defined; the parameterisation changes dynamically as the gate elimination proceeds. This idea helps us simplify the analysis of [19, 34] while also giving us an improved lower bound.

For gate elimination to work, it is important that substituting a variable (or parameter in our case) does not trivialise the function. To ensure this, the hard function we work with must have special properties, depending on the class of variable substitutions allowed. In our case, parameter substitutions will be (for the most part) affine, and the special property of the hard function F we need is that it is an affine disperser. A Boolean function on n bits is an affine disperser for dimension d if it is non-constant on any d -dimensional affine subspace of $\{0, 1\}^n$. An explicit affine disperser for dimension $o(n)$ was given by Ben-Sasson and Kopparty [5], and since then a sequence of improvements [36, 13] has culminated in a construction of an explicit disperser for dimension $O(\log(n))$ by Xin Li [37]. Our argument gives a lower bound of $3.5n - O(d)$ for any Boolean function that is an affine disperser for dimension d , and Theorem 1.1 then follows as a corollary of the construction in [37].

1.3 Previous Work

There are several excellent references on Boolean function complexity. We recommend the books by Wegener [56], Dunne [18] and Jukna [31], and the survey of Boppana and Sipser [9]. On the specific topic of B_2 lower bounds, Golovnev’s thesis [23] and Kulikov’s collection of open problems [32] are well worth studying.

Here we provide a synoptic and self-contained survey of previous work on lower bounds against Boolean circuits. We focus on Boolean circuit lower bounds for Boolean functions. There has also been a lot of work on lower bounds for multi-output functions, and on lower bounds for more restricted models such as formulas and branching programs, for which we advise the reader to consult the references above.

B_2 Lower Bounds: We first discuss work on B_2 lower bounds. In the rest of this discussion, when we refer to “circuits”, we mean B_2 circuits.

Trivially, any Boolean function on n inputs that depends on all its inputs requires circuits of size $n - 1$. The first non-trivial circuit lower bound was shown by Kloss and Malyshev [8], who gave a $2n - O(1)$ size lower bound against B_2 circuits. This lower bound was for the function $\sum_{1 \leq i < j \leq n} x_i x_j$, where addition and multiplication are over the field F_2 . Schnorr [48] gave a $2n - O(1)$ lower bound for the simpler function Mod_3 , where $\text{Mod}_3(x_1, \dots, x_n) = 1$ iff the sum of the inputs is $0 \bmod 3$.

Schnorr’s lower bound holds for a more general class of functions, and is a prototypical example of gate elimination. Consider a circuit C computing the Mod_3 function. We wish to show that we can eliminate 2 gates by setting one of the input variables to a constant, while preserving the property that the resulting circuit computes Mod_3 . Since the restricted function is non-constant even when $n - 3$ variables have been set, it follows that C must have circuit size at least $2n - 6$.

²Those familiar with the theory of exact exponential algorithms will notice a close analogy to the measure-and-conquer paradigm for analysis of exact algorithms for NP-hard problems [21].

Consider any minimal gate g of C in the topological ordering on gates. By minimality, the inputs to g must both be input variables, say x_i and x_j . We claim that at least one of x_i and x_j feeds in to at least one other gate. If not, x_i and x_j affect the output of C only through g . In particular, the output of g must be the same for at least two of the settings $(0, 0)$, $(0, 1)$ and $(1, 1)$ to (x_i, x_j) . But this is a contradiction to C computing Mod_3 as the sub-functions of Mod_3 on these three assignments are all distinct, assuming $n \geq 4$.

Suppose wlog that x_i occurs at least twice in C , and let g_1 and g_2 be gates for which x_i is an input. By setting x_i to 0, both g_1 and g_2 are eliminated (as in general, any gate that depends on at most 1 of its inputs can be eliminated), while the induced sub-function is still Mod_3 on the remaining variables. Thus we can perform gate elimination at least $n - 3$ times to eliminate at least $2n - 6$ gates. Hence the B_2 circuit size of Mod_3 is at least $2n - 6$.

A straightforward case analysis shows that any gate in the B_2 gate set that depends on both its inputs is either of AND type, meaning that it is an AND of its inputs modulo negations at the inputs or output, or of XOR type, meaning that it either computes the XOR function or its complement. The above argument did not distinguish between XOR-type and AND-type gates. Restricting the input to an AND-type gate can have additional benefits - if one of the inputs is set to 0, the gate simplifies to a constant and its successors (which we call its children) are also eliminated. This phenomenon was exploited in subsequent gate elimination arguments.

Stockmeyer [55] showed that the circuit complexity of Mod_3 is at least $2.5n - 3$ by considering sequences of restrictions and showing that at least 2.5 gates are eliminated on average. However, it is known that Mod_3 has circuits of size $3n + O(1)$ [16], and so this method of analysis cannot hope to establish a $3n + \omega(1)$ lower bound.

Paul [41] considered lower bounds for a function closely related to the *indexing* function, and showed a $2.5n - o(n)$ lower bound. By considering a variant of Paul's hard function and building on his ideas, Blum [7] gave a $3n - o(n)$ lower bound for an explicit function.

Blum's lower bound was the record for over 30 years, until a new approach was introduced by Demenkov and Kulikov [17] to show a $3n - o(n)$ lower bound for every affine disperser. This approach was developed further in [19, 20, 34] to obtain cn lower bounds for $c > 3$. A lower bound of $(3 + 1/86)n - o(n)$ was derived in [19, 20] using several new ideas, and this lower bound was then improved to $3.1n - o(n)$ by Li and Yang [34]. All of these lower bounds are for affine dispersers, for which there are now polynomial-time constructions even for dimension $O(\log(n))$ [5, 36, 13, 37]. Our result belongs squarely in this tradition, and we say more about the ideas in Section 1.4.

U_2 Lower Bounds: Recall that the gates in U_2 circuits do not have XOR-type gates - all gates either depend on at most one of the inputs are of AND-type. Because there are restrictions to AND-type gates g which propagate further, i.e., to all children of g , the gate elimination method yields stronger lower bounds on U_2 circuits. In the same paper in which he proved a $2n - O(1)$ size lower bound against B_2 circuits [48], Schnorr also gave a tight $3n - 3$ lower bound against U_2 circuits for the Parity function. Note that this implies that the simulation of size s B_2 circuits by size $3s$ U_2 circuits is tight, as Parity on n inputs does have a trivial B_2 circuit of size $n - 1$.

The gate elimination argument of Schnorr is simple. We consider an arbitrary circuit C for Parity and a minimal gate g in the topological order of gates. Both inputs of g can be assumed to be input variables without loss of generality - let's call them x and y . Either x and y must occur at least twice in the circuit. If not, then there is a constant c such that restricting x to c fixes the gate g and hence the resulting circuit C' does not depend on y . But by assumption C' must compute Parity on $n - 1$ variables (or its complement), which is a

contradiction to the fact that C' does not depend on y .

Assume wlog that x occurs at least twice in the circuit, and that x is an input to gates g and h . At least one of g and h is not the output gate - let's say this is g , without loss of generality. We can assume that g is not an input of any gate h - in such a situation, we can directly rewire the other input of g to h (while perhaps needing to change h to some other AND-type gate h') without increasing the size of C . Thus there is a constant c such that setting x to c fixes g to a constant and eliminates at least one child of g while also eliminating h - at least 3 gates in all. The resulting circuit C' computes Parity or its complement on $n - 1$ variables. By iterating this elimination process for $n - 1$ steps, we get that Parity requires U_2 size $3n - 3$.

This bound is tight for Parity, and therefore further work along this direction considered other classes of functions, while still using gate elimination techniques. Zwick [61] showed a $4n - O(1)$ lower bound for certain symmetric functions. The best lower bounds we know are for k -mixed functions, where a function is called k -mixed if every restriction of a fixed subset of k variable induces a different subfunction on the remaining variables. There are known constructions of $(n - o(n))$ -mixed functions in P . Lachish and Raz [33] gave a $4.5n - o(n)$ lower bound for $(n - o(n))$ -mixed functions, and this was improved to $5n - o(n)$ by Iwama and Morizumi [29]. Amano and Tarui [2] constructed a $(n - o(n))$ -mixed function with circuit complexity $5n + o(n)$, so we require new techniques to get a cn size lower bound for $c > 5$.

The Algorithmic Method and Conditional Lower Bounds: While the gate elimination approach has produced the strongest unconditional results so far, other approaches have been proposed and studied. Particularly intriguing is the *algorithmic method* of Williams [57, 58]. Williams showed that for a circuit class $\mathcal{C}$ with appropriate closure properties, non-trivial satisfiability algorithms for $\mathcal{C}$ imply $\mathcal{C}$ -circuit lower bounds for $\text{NE} = \text{NTIME}(2^{O(n)})$, and then used this connection to obtain unconditional super-polynomial lower bounds in NE against ACC^0 circuits.

Lower bounds in NE seem quite far from explicit lower bounds, but Murray and Williams [39] later scaled down the connection to derive implications for fixed-polynomial $\mathcal{C}$ -circuit lower bounds for NP from better-than-brute-force $\mathcal{C}$ -satisfiability algorithms. In particular, if there is an $\epsilon > 0$ such that satisfiability for $2^{\epsilon n}$ size circuits can be solved in time $2^{(1-\epsilon)n}$, then for each k , NP requires circuits of size n^k .

However, we currently do not have non-trivial satisfiability algorithms even for linear-size circuits. The connections of [57, 39] are not optimal with respect to the circuit size parameter, so it is unclear whether any non-trivial implication for Boolean circuit lower bounds would follow from non-trivial satisfiability algorithms for cn size circuits for constant $c \geq 3$. In the work of [30, 6], the parameters of the connection are optimised, and an explicit constant $d < 2^{1/7}$ is given such that solving 3-SAT in time d^n would imply a $3n$ B_2 circuit size lower bound³ for a Boolean function in E^{NP} .

There are also connections going the other way, from circuit lower bounds to algorithmic results. The second author showed [46] that random restriction arguments used to derive formula size lower bounds could be exploited algorithmically to get the first non-trivial satisfiability algorithms for U_2 formulas. This connection was extended to B_2 formulas by Seto and Tamaki [49]. Chen and Kabanets gave a satisfiability algorithm for B_2 circuits of size $2.5n - o(n)$ and also showed a correlation upper bound for an explicit Boolean function against such circuits [15]. This result was strengthened in [25] to satisfiability algorithms for B_2 circuits of size $2.99n$.

There are other approaches to B_2 lower bounds, which currently only give conditional

³The results of [20, 34] postdate [30].

results. In the same paper in which they give a $(3 + 1/86)n - o(n)$ B_2 size lower bound for an explicit function [20], Find et al. show a $3.11n$ size lower bounds for *quadratic dispersers*, which are stronger objects than affine dispersers, and which currently lack polynomial-time constructions. Golovnev, Kulikov and Williams [26] give an approach to general Boolean circuit lower bounds via low-depth lower bounds. In particular, they show that any Boolean function requiring $2^{n-o(n)}$ depth-3 circuits (over the U_2 basis with unbounded fan-in) with bottom fan-in 16 also requires B_2 circuits of size $3.9n - o(n)$. Currently, no polynomial-time construction of such a function is known.

1.4 Techniques

Gate Elimination: As in all previous unconditional lower bounds for B_2 circuits [8, 48, 55, 41, 7, 17, 20, 34], we use the gate elimination technique. As discussed previously, gate elimination aims to simplify the circuit as much as possible by an appropriate substitution for a variable of the circuit. Here simplification typically means a reduction in the number of gates, but more generally it can mean a reduction in a strategically designed *potential function* W . The potential function is designed so that a reduction in the potential per substitution step implies as large as possible a reduction in the number of gates per substitution step on average over the sequence of substitutions. If the function F we are trying to compute is resilient to the class of substitutions, meaning that each subfunction of F under the sequence of chosen substitutions from the class is non-trivial, then a significant reduction in W per substitution step implies a corresponding lower bound on any circuit computing F . Concretely, if the function F remains non-trivial for $n - o(n)$ substitutions of variables, and the analysis of W gives a reduction of c gates on average per substitution step, we get a lower bound of $cn - o(n)$.

When performing substitutions on B_2 gates, we can disregard gates that do not depend on both of their inputs, as such gates can be eliminated without requiring any substitutions, by merging them with their children. This means that we only need to analyse XOR-type gates and AND-type gates. Fixing an input y to an XOR-type gate eliminates that gate but also all other children of y . If an input y feeds in to an AND-type gate g , we get even more of an advantage, as by setting y to 0 (assuming wlog that the AND-type gate actually computes an AND), we eliminate g , the children of g as well as all other children of y .

Thus, if there is an input which feeds into many gates, we can be assured of significant simplification by setting the input appropriately. This means that in order to get lower bounds of $cn - o(n)$ for an integer c , we only need to consider cases where input variables have low fan-out, i.e., fan-out at most $c - 1$. In principle, this reduces to a case analysis over a finite number of "local" configurations involving a low fan-out input and its children - we need to argue that significant simplification is achievable in each of these cases.

The preliminary analysis above reveals an asymmetry between XOR-type gates and AND-type gates - we would be able to get stronger lower bounds if we could always consider an AND-type gate and substitute an input to it. However, there might not be any input variables feeding directly into AND-type gates at all - all paths from input variables to AND-type gates might pass through XOR-type gates.

This does create an issue if we insist on variable substitutions to constants, but we could allow more general *affine* substitutions which allow us to fix gates feeding into AND-type gates. Networks of XOR-type gates (possibly with multiple outputs) interposing between input variables and AND-type gates compute systems of linear equations in the input variables, and we can use our understanding of linear algebra to manipulate these systems to our advantage. This is a crucial insight in the line of work initiated by Demenkov and Kulikov [17] giving the strongest B_2 lower bounds [20, 34].

Lower Bounds for Affine Dispersers: Demenkov and Kulikov [17] give a simple proof of a $3n - o(n)$ B_2 lower bound for any affine disperser for dimension $o(n)$. Recall that an affine disperser for dimension d is a Boolean function that is non-constant on every affine space of dimension d . The explicitness question for affine dispersers has been very well studied [5, 36, 13, 37] and there is even a polynomial-time construction known for dimension $O(\log(n))$ [37].

Suppose that there is a B_2 circuit C' computing an affine disperser F for dimension $d = o(n)$. Demenkov and Kulikov define a potential function W for B_2 circuits C : $W(C) = s(C) + i(C)$, where $s(C)$ is the number of internal gates in the circuit and $i(C)$ is the number of inputs to the circuit. Crucially, we allow any gate g that computes an affine function of the input variables to be re-designated as an input to the circuit. Initially, $W = s + n$, where s is the size of C' and n is the number of input variables. We assume without loss of generality that the circuit has no redundant gates or inputs, i.e., gates that do not have a path to the output or inputs with out-degree 0. These can be removed as and when they arise in the gate elimination process, without affecting the functionality of the circuit.

The idea is to pick a *minimal* AND-gate g , i.e., an AND-gate that is minimal in the topological ordering of AND-gates, and fix one of its inputs. Such an AND-gate must exist initially as C does not compute an affine function, and it is not the output gate as otherwise C would be constant on an affine space of dimension $n - 1$. The inputs of such an AND-gate must both be affine functions of input variables. Both of these can be re-designated as inputs to the circuit without changing the potential W . Call these inputs x and y .

If x and y both have out-degree 1, we can fix x to a constant to eliminate g and at least one successor of g , since g is not an output gate. Also y does not feed into any gates of the circuit once x is fixed, and so it can be eliminated too. Thus 2 inputs and 2 gates have been removed, and W decreases by 4.

Suppose instead that one of the inputs, say x has out-degree at least 2, and feeds into gates g and g' , of which g is not an output gate. We can assume wlog that g is not an input to g' - if this were the case, we could replace g' with an AND-type gate that has x and y as inputs. Fixing x to an appropriate constant eliminates g , g' and a successor of g - these are all distinct gates. Thus 1 input and 3 gates have been removed, and W reduces by 4.

We have not specified what it means to “fix” an input x which is originally an internal gate of the circuit. When we speak of such a fixing, we simply mean that we consider the subset of the original input space satisfying all the fixing conditions. Since each fixing is to an affine function, we restrict attention throughout the argument to affine subspaces of the original input space. After k fixings, the potential W goes down by at least $4k$, and the resulting subfunction is non-trivial as long as $n - k > d$. Since the initial potential is $s + n$ and we can apply the fixing argument $n - d$ times, we get that $s + n \geq 4(n - d)$, implying that $s \geq 3n - 4d$. Vadhan and Williams (cited as personal communication in [17]) observed that this is essentially tight, as the Inner Product function has B_2 circuit size $n - 1$ and is an affine disperser for $d = n/2 + 1$.

However the tightness of the argument does not preclude obtaining a lower bound of the form $cn - c'd$ for $c > 3$ and constant c' . Indeed, this was achieved by Find, Golovnev, Hirsch and Kulikov [20], who obtained as a consequence a $(3 + 1/86)n - o(n)$ lower bound for any affine disperser F for dimension $o(n)$.

Several new ideas were introduced in [20], which are also crucial in our argument. While the argument in [17] used fixings of affine functions occurring within the circuit, [20] use *affine substitutions*, which are substitutions of an input variable x by an affine function of the remaining variables. Such a substitution can be implemented as long as the corresponding affine

function is computed by some gate of the current circuit. The notion of a circuit is generalized in [20] to *fair semi-circuits*, where the original circuit C is factored into an acyclic part which may contain AND-type gates, and a potentially cyclic part containing only XOR-type gates. The crucial observation here is that cyclic circuits make sense computationally as long as any assignment to the input variables yields a unique assignment of the gate variables. Allowing affine substitutions and considering a fair semi-circuit representation of the circuit makes it easier to reduce to the minimal-AND case discussed above in the context of the argument in [17]. In our argument too, we use affine substitutions and a fair semi-circuit representation.

There are a couple of other important new ideas in [20]. They define the notion of a *troubled gate*, which is an AND-type gate g of out-degree 1 with inputs x and y of out-degree 2. Such a gate poses a bottleneck to simplifying the circuit sufficiently to beat the $3n - o(n)$ lower bound. In order to deal with such gates, they consider a potential function that incorporates the number of troubled gates, and allows to control the production of troubled gates during gate elimination. In addition, a particularly problematic kind of troubled gate is what we call a *bottleneck configuration* in this paper. A bottleneck configuration is a troubled gate g (with inputs x and y) which feeds into an XOR-type gate h whose other input is an input variable z of out-degree 1. To deal with bottleneck configurations, a limited kind of *quadratic substitution* to variables is allowed in [20]. Namely, they allow setting z to a quadratic function of x and y in a way that eliminates g , h and any gates that h feeds into.

However, quadratic substitutions are dangerous because they do not in general preserve the non-triviality of the function F we are computing, which we only assume is an affine disperser⁴. Therefore, they are only allowed in a limited way - when a quadratic substitution is made in which variables x and y are paired (i.e., occur in a quadratic term), then x and y are not subject to arbitrary affine substitutions (which might introduce many more quadratic terms) or quadratic substitutions (which might introduce terms of higher degree in the remaining variables). Instead, we insist that any fixing of x, y satisfies $x = y + c$ for some constant c , which causes the pair to disappear. As a consequence of these constraints, quadratic terms always involve disjoint pairs of variables when they arise. It can be argued that affine dispersers remain non-trivial under such limited quadratic substitutions.

The final argument in [20] is a complicated case analysis where a lot of effort is taken to control the number of troubled configurations. Li and Yang [34] introduced several additional ideas to control the number of troubled configurations in better ways, and showed that these ideas could be leveraged to get a $3.1n - o(n)$ lower bound.

The proof in this paper: In this paper too, we prove a lower bound for any affine disperser F using gate elimination. The main novelty here is that we do not need to control the number of troubled configurations. The key to this is a conceptual change: instead of considering substitutions on the space of input variables as done in [20, 34], we consider substitutions on a space of *independent parameters* which can be modified as the gate elimination argument proceeds. The dependence of input variables on parameters is defined by an injective map ϕ from the parameter space to the variable space. Each component of ϕ is an affine function plus quadratic terms on disjoint pairs of parameters. We begin with ϕ as the identity map, and update it as the gate elimination process goes on and the number of parameters decreases.

In addition to affine substitutions and limited quadratic substitutions on parameters (as in [20]), we define *affine reparameterisation* (by introducing a new parameter which is an affine function of the old ones, and removing an old one, thus preserving the number of parameters)

⁴There are currently no good explicit constructions of quadratic dispersers, which are dispersers for solution spaces for sets of quadratic equations.

and *quadratic reparameterisation* (which again preserves the number of parameters while doing a quadratic shift).

Affine reparameterisation facilitates the reduction to the case of a minimal AND with parameters as inputs, and quadratic reparameterisation allows us to handle some kinds of troubled gates without actually doing a substitution. Indeed, our improvements come from arguing that in some cases such as bottleneck configurations where gate elimination does not seem to give significant benefits, we can avoid substitutions while eliminating the bottleneck.

As in [20], we use affine and restricted quadratic substitutions, and represent the circuit at any stage of the gate elimination process as a fair semi-circuit. Our fair semi-circuit representation is a bit more general than theirs: rather than decomposing into a cyclic circuit of XOR-gates and an acyclic part, we decompose into an affine system consisting entirely of linear equations and an acyclic part. The representation of the affine system does not track inputs or outputs of gates - instead, each variable in an equation is either a parameter or an "auxiliary variable". We require that each equation has at most 3 variables (corresponding to the fact that every XOR-type gate has two inputs and one output), and also that any assignment to parameters gives a unique assignment to auxiliary variables. This sufficiently captures the semantics of cyclic circuits for our purposes.

As in several previous works, we use a potential function W for the analysis of the gate elimination process, however this potential function is defined in a different way than in preceding works. W is a weighted linear sum of four quantities: the number of gates s in the current circuit (defined to be the number of gates in the acyclic part plus the number of equations), the number m of independent parameters, the number q of pairs of parameters (corresponding to the number of quadratic substitutions at the current point in time) and the sum of weights of unpaired parameters, where the weight of an unpaired parameter is defined based on its number of occurrences in the circuit. To optimise the analysis, we set $W = 2s + 3q + \sum_x w(x)$, where the sum is over unpaired parameters x , and the weight of a parameter is defined to be 6 if it is unpaired with a single occurrence in the circuit, 7 if unpaired with more than one occurrence, and 0 otherwise.

Our gate elimination process progressively reduces the sum $s + m$ of the current circuit size and number of parameters, with a significant decrease in W per parameter substitution, while preserving non-triviality of the corresponding sub-function F' of our affine disperser F . It turns out to be easier to track the change in potential as a function of $a = m - q$, where q is the number of pairs of parameters used in quadratic substitutions. We define an efficient algorithmic process which reduces a by 1 or 2 at each step, while reducing W by at least 14 times the change in a . We are able to argue non-triviality of the resulting sub-function F' as long as $a > d$, and if we choose an affine disperser F for dimension $O(\log(n))$, this means that we can reduce W by $14(n - O(\log(n)))$ before the gate elimination process terminates. Since W was at most $2s + 7n$ to begin with, where s is the size of the original circuit C computing F , we have that C must have size at least $3.5n - O(\log(n))$.

The core of the gate elimination process is to reduce to the case where we have a minimal AND-gate with parameters feeding into it. We can find substitutions to achieve the required decrease in potential except in the case of a bottleneck configuration, where we use quadratic reparameterisation.

Stepping back, there are two broader points we'd like to make. First, though gate elimination has been used in all previous unconditional lower bounds for B_2 circuits, progress has typically been achieved by adopting a more *general* view of gate elimination. Initially, gate elimination was framed purely in terms of elimination of gates by substituting constants for variables. Later potential functions were defined to analyse amortised savings, and more general substitutions for variables were considered. The proof in this paper generalises gate

elimination further by considering substitutions to a dynamically evolving parameter space rather than the space of variables, and this generalisation might have other applications.

Second, the analysis in this paper not just achieves better parameters, but simplifies the case analysis considerably compared to [20, 34]. While these previous works need to consider dozens of cases, the case analysis here only requires 10 cases and fits into a few pages. For maximum clarity, we have given all details of these cases, including routine calculations - the proof could be presented much more succinctly for readers who are familiar with gate elimination arguments.

1.5 Barrier Results and their Relevance

Given that progress on unconditional B_2 lower bounds has been very slow over several decades, it is worthwhile to investigate barriers on current techniques.

General barriers: The relativization barrier [3] is not relevant to questions about fixed polynomial circuit lower bounds, because known techniques do take advantage of the structure of the gate set, and do not work for circuits with oracle gates. The natural proofs barrier of Razborov and Rudich [44] is potentially more relevant: under standard cryptographic assumptions, i.e., the existence of sub-exponentially secure one-way functions, the barrier shows that there are no “natural” proofs of super-polynomial size lower bounds against Boolean circuits. Here a natural proof is a property of Boolean functions (given by their truth tables), that is constructive (i.e., decidable in polynomial time), useful (i.e., no function satisfying the property has small circuits) and dense (i.e., a noticeable fraction of Boolean functions satisfy the property).

It is unclear if the natural proofs barrier says anything interesting about lower bound arguments against linear-size circuits. Miles and Viola [38] give candidate constructions of exponentially secure one-way functions with circuit size $n(\log(n))^{O(1)}$. Under the assumption that these constructions are secure, there is a constant c such that there are no natural proofs against circuit size $n \log(n)^c$. However, no good candidates are known for such constructions with circuit size $O(n)$.

It is also unclear if gate elimination proofs yield natural properties. The lower bounds of [19, 34] and this paper work for any affine disperser. It is true that the property of being an affine disperser is dense, but it is unclear if it is constructive. The best known algorithm for checking if a Boolean function on n bits (given by its truth table of size $N = 2^n$) is an affine disperser for dimension $O(\log(n))$ runs in time $N^{O(\log \log(N))}$, by enumerating all affine subspaces of dimension $O(\log(n))$ and checking that the function is non-constant on all of them.

Barriers for gate elimination: Perhaps we could at least find limitations on gate elimination as a lower bound technique? Intuitively, analysing “local” configurations of circuits (i.e., analysing all gates an input feeds into) should not be sufficient for proving super-linear lower bounds for B_2 circuits. Golovnev, Hirsch, Knop and Kulikov [24] formalise this intuition for certain versions of gate elimination. In particular, they show that it is not possible to prove a lower bound greater than $5n$ by simply substituting variables as arbitrary functions of the remaining variables and arguing that more than 5 gates are eliminated by each substitution. They then give a more general argument for lower bounds shown via potential functions, though here the limitation on the largest provable lower bound depends on properties of the potential function.

It remains unclear what limitations there are on the power of gate elimination as generalised in this paper. Note that we do not just allow substitutions to variables, but rather to underlying parameters which are themselves defined during the gate elimination process.

Barriers on lower bounds for affine dispersers: A way of showing limitations on the specific methods of [20, 34] and this paper would be to show that the hard functions considered in these works are actually computable in linear size. It is currently unknown if affine dispersers are computable by B_2 circuits in linear size, and if so, what the best linear-size upper bound is.

1.6 AI Disclosure Statement

Briefly, the proof of the main result was elicited from AI tools, and our contribution is mainly in verifying, interpreting, simplifying and presenting the proof.

Initial prompts to AI tools to produce a proof of a lower bound better than $3.1n - o(n)$ were unsuccessful, but after several attempts, the tools eventually came up with an improved bound of $3.125n - o(n)$. With many more rounds of interaction, the significantly better lower bound of $3.5n - o(n)$ was obtained. The original proof of this lower bound was opaque and rather complicated - the proof was then simplified using a further back-and-forth with the AI tools.

The paper was rewritten from scratch using just the proof skeleton produced by AI tools as a guide. We have kept the following three principles in mind when writing the paper. First, we wanted to present the intellectual genealogy of this work as accurately as possible, and give adequate credit to previous works. Second, we wanted to make the proof as simple as possible, and understandable for anyone with a basic complexity-theoretic background. Third, we wanted to present the material in a way that makes the crucial ideas as transparent as possible, and enables further progress and understanding of the limitations of current techniques.

1.7 Organisation of the Paper

In Section 2, we define our notation and explain the basic concepts we use. In Section 3, we explain how deletions and reparameterisations work during the gate elimination process. In Section 4, we explain how our gate elimination algorithm works, and analyse it to obtain Theorem 1.1.

2 Preliminaries

2.1 Circuits and Notation

We use the standard model of Boolean circuits over the full binary basis, see Wegener [56] and Jukna [31] for background, and the papers by Find et al. [19] and Li & Yang [34] for the conventions closest to ours. A *fan-in two Boolean circuit* is an acyclic directed graph in which every vertex has indegree zero or two, and the edges entering a vertex are ordered. Vertices of indegree zero are called *input vertices* and are labelled by variables $X_1, \dots, X_n$ or constants 0, 1. Vertices of indegree two are called *gates*. The *fan-out* of a vertex is its outdegree. We call the edges *wires*. Both wires entering a gate may exit from the same vertex. The fan-out of a vertex is unrestricted. Each gate is labelled by a Boolean function $g : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$; the set of all sixteen such functions is denoted by B_2 . One gate of outdegree zero is designated as the *output*. The *size* of a circuit C , denoted by $|C|$, is its number of gates. 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 either input variables or the results of the previous instructions.

We call the functions labelling gates *operations* in order to distinguish them from functions of n variables computed in the gates. A gate is *constant* if its operation ignores both inputs, and *unary* if its operation depends on exactly one of the inputs, in which case the gate computes that input or its negation. Of the sixteen operations, two are constant, four are unary, and the remaining ten depend on both inputs. The remaining ten operations consist of two *XOR-type* and eight *AND-type* operations: up to negating the inputs and the output, every XOR-type operation is XOR and every AND-type operation is AND. We call a gate constant, unary, XOR-type, or AND-type according to its operation.

We identify $\{0, 1\}$ with the field $\mathbb{F}_2$ of two elements, so that addition is XOR, multiplication is AND, and negation is addition of 1. Every operation $h: \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$ has a unique multilinear expression $h(t_1, t_2) = c + b_1 t_1 + b_2 t_2 + b_{12} t_1 t_2$, where $c, b_1, b_2, b_{12} \in \mathbb{F}_2$. In particular, the XOR-type operations have the form $t_1 + t_2 + c$, and the AND-type operations have the form $(t_1 + \alpha)(t_2 + \beta) + \gamma$, where $c, \alpha, \beta, \gamma \in \mathbb{F}_2$. Thus, XOR-type operations are polynomials of degree one, and AND-type operations are polynomials of degree two. More generally, an *affine operation* has the form $b_1 t_1 + b_2 t_2 + c$, where either coefficient $b_1, b_2 \in \mathbb{F}_2$ may vanish. We call all gates labelled with an affine operation as *affine gates*.

A *literal* of a value t is either t or $t + 1$. We allow the output to be a constant or a literal, and we allow negations on wires at no cost by absorbing them into the operations of the gates that read those wires.

2.2 Parameters

We write $\vec{X} = (X_1, \dots, X_n)$ to denote the input variables of the affine disperser $f: \mathbb{F}_2^n \rightarrow \mathbb{F}_2$. During gate elimination, we maintain an injective map $\Phi: \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$, whose variables $\vec{u} = (u_1, \dots, u_m)$ we call *parameters*, together with a circuit C that reads the parameters (in place of the original inputs), which on evaluation at the parameter $\vec{u}$, computes the value $f(\Phi(\vec{u}))$. In other words, $\Phi(\vec{u})$ is the input to f , and the main invariant of gate elimination is that, at every step, $C(\vec{u}) = f(\Phi(\vec{u}))$ for every parameter assignment $\vec{u}$. By injectivity, distinct parameter assignments u describe distinct assignments $\vec{X} = \Phi(\vec{u})$ of f . Initially, Φ is the identity and $m = n$, so the parameters $\vec{u}$ are just the original inputs $\vec{X}$.

More precisely, we maintain the map $\Phi: \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$ in the form

$$\Phi(\vec{u}) = \vec{b} + L\vec{u} + \sum_{\{p,r\} \in P} \vec{c}_{p,r} pr, \quad (1)$$

where $\vec{b} \in \mathbb{F}_2^n$, $L \in \mathbb{F}_2^{n \times m}$, P is a set of disjoint pairs of parameters, and $c_{pr} \in \mathbb{F}_2$ for each pair $\{p, r\} \in P$. We say that a parameter that belongs to a pair in P is *paired*, the other member of its pair is its *partner*, and all other parameters are *unpaired*. A pair in P records that its two parameters may occur together in a quadratic term. Since the term pr becomes affine once either p or r is fixed to a constant, or p is identified with a literal of r , the pair $\{p, r\}$ provides only one degree of freedom. Therefore, we consider p and r together during gate elimination, rather than working with only one member of the pair. Paired parameters may also occur in the affine part $L\vec{u}$, and we allow $c_{pr} = 0$.

2.3 Fair semi-circuits and the affine system

We work with *fair semi-circuits*, which are a generalization of circuits introduced by Find et al. [19]. A fair semi-circuit consists of an affine system of equations followed by an acyclic part that is an ordinary circuit whose gates read the parameters and the values determined by the affine system.

Fair semi-circuits. Let $\vec{u} = (u_1, \dots, u_m)$ be the parameters. A *fair semi-circuit* on $\vec{u}$ is the composition of an affine system and an acyclic part, together with an output, described below.

- (i) The *affine system* consists of $r \geq 0$ equations over $\mathbb{F}_2$ in the parameters and in r further variables $\vec{v} = (v_1, \dots, v_r)$, which we call *auxiliary variables*. We write it as $A\vec{v} = N\vec{u} + \vec{c}$, where $A \in \mathbb{F}_2^{r \times r}$ and $N \in \mathbb{F}_2^{r \times m}$ contain the coefficients of the auxiliary variables and of the parameters, respectively, and $c \in \mathbb{F}_2^r$ contains all the constant terms. We call A the *auxiliary matrix*, and N the *parameter matrix*. Each equation contains at most three nonconstant terms, that is, every row of the matrix $(A \mid N)$ has at most three nonzero entries. Moreover, we require *fairness*, that the affine system has a unique solution v for every parameter assignment u , this is equivalent to A being invertible.
- (ii) The *acyclic part* is an ordinary circuit whose gates are listed in a topological order. Each gate is labelled by an operation from B_2 , and each of its two inputs is a constant, a parameter, an auxiliary variable, or an earlier gate. There are no wires going back from the acyclic part to the affine system.
- (iii) The *output* is a constant or a literal of a parameter, an auxiliary variable, or a gate.

Note that no variable of an equation in the affine system of a fair semi-circuit is designated as an output, this will later allow us to exchange the roles of auxiliary variables and parameters.

We refer to parameters, auxiliary variables, and gates as *internal values*. By the fairness condition, every parameter assignment $\vec{u}$ extends uniquely to an assignment to all internal values that satisfies every equation in the affine system and gives every gate of the acyclic part the value of its operation on its inputs. In particular, every auxiliary variable is an affine function of the parameters, which is given by $\vec{v} = A^{-1}N\vec{u} + A^{-1}\vec{c}$. Figure 1 shows an example of a fair semi-circuit.

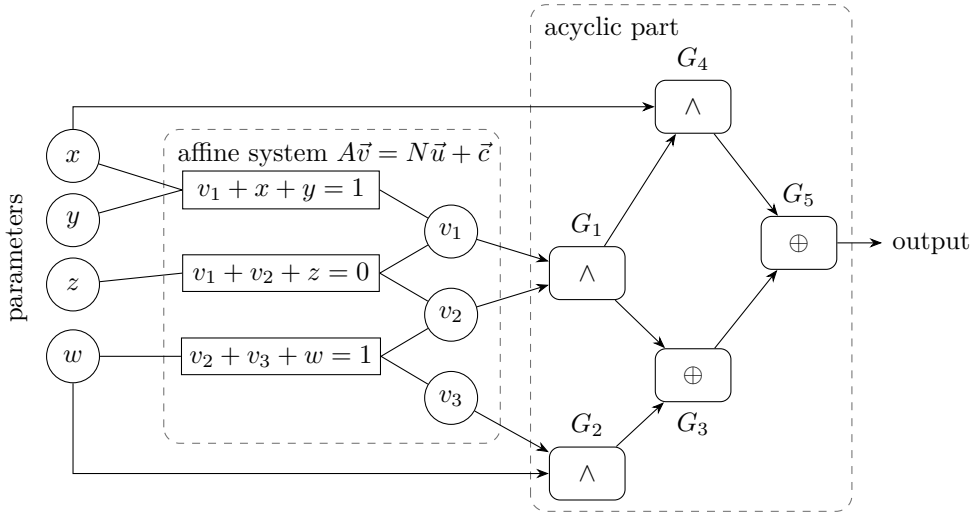


Figure 1: A fair semi-circuit. The affine system consists of $r = 3$ equations over $\mathbb{F}_2$ in the parameters $\vec{u} = (x, y, z, w)$ and the auxiliary variables $\vec{v} = (v_1, v_2, v_3)$, and each equation contains three nonconstant terms. The auxiliary matrix A is lower triangular with ones on the diagonal, and hence invertible, so the fairness condition holds. The acyclic part consists of the gates $G_1 = v_1 v_2$, $G_2 = v_3 w$, $G_3 = G_1 + G_2$, $G_4 = G_3 x$, and $G_5 = G_3 + G_4$, listed in a topological order, and the output is the gate G_5 .

Since these equations are solved simultaneously rather than evaluated one at a time, they may depend on each other cyclically. We allow cyclic dependencies, while requiring that the

system has a unique solution. For example, the system

$$v_1 + v_2 = x, \quad v_2 + v_3 = y, \quad v_1 + v_2 + v_3 = 0 \quad (2)$$

over $\mathbb{F}_2$ (where x, y are parameters), has the auxiliary matrix

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

and a unique solution $(v_1, v_2, v_3) = (y, x + y, x)$ for the auxiliary variables. Rewriting the system as $v_1 = x + v_2$, $v_2 = y + v_3$, and $v_3 = v_1 + v_2$ creates a cycle in the dependency graph for the auxiliary variables, yet the matrix is invertible, refer to Figure 2 for an illustration.

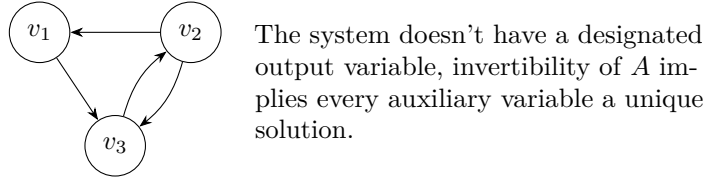


Figure 2: Cyclic dependencies in the affine system allow unique solvability.

Throughout gate elimination, s denotes the *size* of the current fair semi-circuit, that is, the number r of equations plus the number of gates in the acyclic part. Initially, the affine system is empty and the acyclic part is the original circuit C , so that $s = |C|$. As the algorithm proceeds, affine gates are moved from the acyclic part into the affine system, as described in more detail below.

Moving new gates into the affine system. Suppose that a gate J of the acyclic part is labelled by an affine operation $h(t_1, t_2) = b_1 t_1 + b_2 t_2 + c$, where $b_1, b_2, c \in \mathbb{F}_2$, and that each of its inputs t_1, t_2 is a constant, a parameter, or an auxiliary variable. Replace the gate J by the equation

$$J + b_1 t_1 + b_2 t_2 = c$$

and regard J as a new auxiliary variable. Observe that this new equation has at most three nonconstant terms, and that the auxiliary matrix of the new affine system has the form

$$A' := \begin{pmatrix} A & 0 \\ * & 1 \end{pmatrix},$$

which is invertible. Moreover, for every parameter assignment, the solution of the old system extends uniquely to a solution of the new system, in which auxiliary variable J takes the value previously computed by the gate J . Every use of J now refers to the auxiliary variable J , so the function computed by the fair semi-circuit is unchanged. Since one gate has been replaced by one equation, the size s of the fair semi-circuit is preserved. Applying this step to the gates in topological order, we move every gate with an affine operation and no AND-type ancestor into the affine system, while all gates with an affine operation and an AND-type ancestor stay in the acyclic part.

Comparison with Find et al [19]. Our fair semi-circuits differ only slightly from those of Find et al. [19], but the difference allows us to perform affine substitutions without changing the circuit. In their setting, the acyclic part is also an ordinary circuit, but the role of the

affine system is played by a cyclic circuit of affine gates. Written as a system of equations, this has the form $A\vec{v} = N\vec{u} + \vec{c}$ with an invertible matrix A , where the unknowns are the values of its gates and the input variables play the role of the parameters. However, their equations have additional structure: each equation in the cyclic part also has a designated output, that is, each equation corresponds to a gate and defines the value of this gate in terms of its two inputs. Find et al. also record the substitutions made so far by a map from their free variables to all input variables, similar to our map Φ . However, their free variables are the input variables themselves, and their circuits read the input variables directly. Therefore, an affine substitution for a free variable has to be implemented directly in the circuit, which Find et al. do by rewiring its linear part without adding new gates, and this rewiring is the reason why cycles may appear in their circuits. In our setting, the circuit reads the parameters $\vec{u}$, and the input to f is $\Phi(\vec{u})$, so the map Φ “decouples” the circuit from the input variables, which are related to the circuit only through the main invariant $C(\vec{u}) = f(\Phi(\vec{u}))$. In our setting, no variable of an equation is designated as an output. This allows us to exchange the role of an auxiliary variable and a parameter, and thus perform an affine substitution only in the map Φ , leaving the circuit unchanged upto renaming, as explained in detail in Section 3.2.

In what follows, we abuse notation by using the word “circuit” to mean a fair semicircuit.

2.4 States and substitutions

At every step of gate elimination, the *state* consists of the current circuit C , the injective map Φ , and the set of disjoint pairs of parameters P . We keep track of it through four quantities: the size $s = |C|$ of the current circuit, the number m of parameters, the number $q = |P|$ of pairs, and the difference $a = m - q$. Often we allow parameters to remain in Φ even when C no longer reads them, so m counts them as well.

Changes of parameters and restrictions. We simplify the circuit C using four kinds of change to the parameters in Φ : two reparameterisations, (AR) and (QR), and two restrictions, (UF) and (PF). We describe them below, using $\vec{u}_{-x}$ to denote all parameters other than x , and w to denote a fresh parameter not occurring in the map Φ .

Affine reparameterisation (AR). For an unpaired parameter x and an affine map φ , substitute $x \leftarrow w + \varphi(\vec{u}_{-x})$. This reparameterisation does not change the values m, q, a , as all the paired parameters are unaffected, and the new parameter w replaces the old parameter x .

Quadratic reparameterisation (QR). For distinct unpaired parameters x, y, z and constants $\alpha, \beta, \gamma \in \mathbb{F}_2$, substitute

$$z \leftarrow w + (x + \alpha)(y + \beta) + \gamma, \quad (3)$$

and add the pair $\{x, y\}$ to P . This reparameterisation does not change m , increases q by one as the pair $\{x, y\}$ is added to P . Hence, a decreases by one.

Fixing an unpaired parameter (UF). For an unpaired parameter x and an affine map ψ , impose $x = \psi(\vec{u}_{-x})$. Since ψ may be the constant map, this includes fixing an unpaired parameter x to a constant. This restriction decreases m by one and leaves q unchanged, so a decreases by one.

Fixing a paired parameter (PF). For a pair $\{p, r\} \in P$, either fix p to a constant or impose $p = r + c$ for some $c \in \mathbb{F}_2$, and then remove p and the pair $\{p, r\}$. The same can be done for r , the other parameter of the pair. This restriction decreases both m and q by one, so a is unchanged.

Each operation replaces the map Φ by $\Phi \circ \tau$, where τ expresses the old parameters in terms of the new ones. In (AR) and (QR), τ is a bijection, whose inverse is obtained by solving the substitution for w , so $\text{im } \Phi$ is unchanged. In (UF) and (PF), τ fixes one parameter as a function of the others, so it is injective and $\text{im } \Phi$ can only shrink. In all four cases, $\Phi \circ \tau$ is injective as both Φ and τ are injective, and it again has the form in (1).

We make the same change in the circuit C , so that the new circuit computes $C \circ \tau = f \circ (\Phi \circ \tau)$, and the main invariant is preserved. The gate elimination algorithm uses (UF) only to fix a parameter to a constant or to impose $x = y + c$. Thus, in each application of (UF) or (PF), each occurrence of the restricted parameter becomes a constant, or a literal of another parameter. This adds no new gates to the circuit, and all the gates and equations that it simplifies are counted in s until they are deleted. Substituting the expressions for the relevant parameter directly into the circuit in (AR) or (QR) could add new gates, so these are implemented differently, which we explain in more detail in Section 3.2. Each application of (AR) in gate elimination leaves the circuit unchanged up to renaming, and (QR) is used only where it allows two gates to be deleted.

2.5 Affine dispersers

An affine subspace of $\mathbb{F}_2^n$ is a translate $b + V$ of a linear subspace $V \subseteq \mathbb{F}_2^n$, for some $b \in \mathbb{F}_2^n$.

Definition 2.1 (Affine dispersers). *A function $f: \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is an affine disperser for dimension d if it is nonconstant on every affine subspace of dimension d .*

Explicit constructions of affine dispersers over $\mathbb{F}_2$ have been studied extensively. The first constructions gave polynomial-time computable affine dispersers for linear dimension $d = \delta n$ for any constant $\delta > 0$ [4, 10], and later ones reached sublinear dimension [5, 60, 35, 50, 36, 13, 14]. The best known construction was obtained by Li [37] for dimension $d = O(\log n)$, which is optimal up to the constant factor.

The image of the injective map Φ in a state need not be affine, but the next proposition shows that it contains an affine subspace of dimension a .

Proposition 2.2. *The image of every injective map of the form (1) contains an affine subspace of dimension $a = m - q$. Consequently, $f \circ \Phi$ is nonconstant whenever $a \geq d$.*

Proof. We choose one parameter from each pair in P and fix its value to a constant. Since the pairs are disjoint, this fixes exactly q parameters, and every quadratic term in Φ then becomes affine in the remaining $m - q$ parameters. Moreover, this restricted map is injective, as Φ is. Therefore, its image is an affine subspace of dimension $a = m - q$. If $a \geq d$, then f is nonconstant on $\text{im } \Phi$, as f is an affine disperser of dimension d . $\square$

2.6 Degrees, the potential, and valid simplifications

Parents, children, and isolated parameters. We say that the *parents* of a gate are its inputs, which may be constants or internal values, and that the gate is a *child* of each internal value it reads. Similarly, we say that an equation is a child of each internal value that appears in it with a nonzero coefficient, and we call the internal values appearing in an equation the parents of that equation. If a gate reads the same internal value twice, it counts as one child of that internal value. In particular, the children of a parameter x are exactly the equations that mention x and the gates that read x .

The *degree* $\deg(t)$ of an internal value t is its number of distinct children. We say that a parameter is *unused* if its degree is zero and *isolated* if its degree is one. The notion of degree

is syntactic: it counts only direct occurrences. For instance, once the affine system is solved, an auxiliary variable v may depend on a parameter x even if no equation contains both v and x , such a dependence does not contribute to $\deg(x)$.

Rerouting. When a gate or auxiliary variable becomes a constant or a literal of another internal value, we replace all of its occurrences by that constant or literal, we call this *rerouting*. Rerouting can increase the degree of a parameter that was a parent of a replaced gate, in particular, it can turn an isolated parameter into one of degree at least two.

The potential function. We define the *weight* of an unpaired parameter x by

$$\text{wt}(x) = \begin{cases} 0, & \deg(x) = 0, \\ 6, & \deg(x) = 1, \\ 7, & \deg(x) \geq 2, \end{cases} \quad (4)$$

and the *potential* by

$$W = 2s + 3q + \sum_{x \text{ unpaired}} \text{wt}(x). \quad (5)$$

Note that paired parameters only contribute through the $3q$ term in the potential, they have no individual weight. The potential is always nonnegative.

For any quantity X , including m, s, q, a, W , we write X for its value before an operation and X' for its value afterwards, and we set

$$\Delta X = X' - X. \quad (6)$$

In the analysis that follows, we may group several operations into one step, Δ compares the value of the quantity at the beginning and end of the whole step.

Preserving the representation. We now briefly explain why the various simplifications introduced above preserve the two main invariants during gate elimination: every equation has at most three nonconstant terms, and the auxiliary matrix is invertible.

For any parameter x , fixing it to a constant c_0 deletes the column corresponding to x in the parameter matrix N , updates the constant vector $\vec{c}$, and does not change the auxiliary matrix A . Similarly, for distinct parameters x and y , setting $x = y + c_0$ does not change the auxiliary matrix A , and does not increase the number of nonconstant terms in any equation.

A general affine substitution for a parameter into the system $A\vec{v} = N\vec{u} + \vec{c}$ could expand one term into many, and thus violate the bound of three nonconstant terms per equation. Therefore, we implement (AR) by exchanging the roles of a parameter and an auxiliary variable, and only performing the affine substitution in the map Φ . In more detail, suppose that an auxiliary variable Q has the solved expression $Q = x + \varphi(\vec{u}_{-x})$ in terms of the parameters, where x is unpaired and φ is affine. We substitute $x \leftarrow w + \varphi(\vec{u}_{-x})$ in Φ , rename Q as the new parameter w , and regard x as an auxiliary variable. This does not change the set of equations, and only changes the role of x and Q in the system. More concretely, we exchange the column corresponding to x in N with the column corresponding to Q in A in the affine system. Hence, every equation keeps exactly the same terms. To see that the new auxiliary matrix A' is invertible, let j be the column index of Q , let N_x be the column of x in N , and let $t = A^{-1}N_x$. The coefficient of x in the expression $Q = x + \varphi(\vec{u}_{-x})$ is $t_j = 1$ (as Q depends on parameter x). Since A' is obtained from A by replacing its j th column with N_x , we have $A' = AB$, where B is the identity matrix with its j th column replaced by t . As $\det B = t_j = 1$, the matrix A' is invertible.

Similarly, we apply (QR) only to a isolated parameter z whose single use is as an input of an XOR-type gate D , where the other input of D computes $(x + \alpha)(y + \beta) + c_0$, for parameters x and y . After an application of (QR), D computes a literal of w , so we reroute D to that literal of w and delete it. As z does not occur in any equation, the affine system is unchanged.

Finally, we show that rerouting preserves both invariants. First, note that rerouting a gate in the acyclic part does not change the affine system, as there are no wires going back from the acyclic part to the affine system. Now, let v be an auxiliary variable that occurs in an equation with at most two nonconstant terms, which we can write as $v = U + c_0$, where U is a parameter or an auxiliary variable. To reroute v , we replace every occurrence of v by $U + c_0$, and delete v together with the equation $v = U + c_0$. Observe that replacing a term by a constant, or by a literal of another single term, cannot increase the number of nonconstant terms in any equation. To see that the new auxiliary matrix is invertible, let v be the k th auxiliary variable, and let $v = U + c_0$ be the i th equation. Replacing v by $U + c_0$ in another equation is the same as adding the i th row of $(A \mid N \mid c)$ to the row corresponding to that equation. Restricted to A , these are elementary row operations, so they preserve $\det A$. After substituting $v = U + c_0$ in every valid equation, the k th column of A has a single nonzero entry in row i . Expanding the determinant along this column, we see that deleting row i and column k from A leaves a matrix whose determinant equals $\det A$ up to sign. Hence, the auxiliary matrix of the new system is invertible.

The same argument applies when we delete an auxiliary variable v whose only use is in a single equation in the affine system, together with the equation v occurs in. Every other equation is unchanged by this deletion, so only invertibility of auxiliary matrix remains to be verified. The column of v in A already has a single nonzero entry, so the determinant expansion above applies directly, showing that the auxiliary matrix of the new system is invertible.

The next lemma shows that, as long as $a > d$, the output cannot take certain simple forms. We use it later to rule out several degenerate cases in the gate elimination algorithm.

Lemma 2.3. *Let $a > d$. Then the output of the circuit cannot be an affine function of the parameters, or a literal of an AND-type gate whose two parents are unpaired parameters.*

Proof. First, note that a constant output would contradict Proposition 2.2. Suppose that the output is a nonconstant affine function of the parameters, choose a parameter x with nonzero coefficient. If x is a paired parameter, let y denote its partner. Fix y to a constant, this does not change $a = m - q$, and makes x unpaired. Now that x is unpaired, substitute for x an affine expression in terms of the other parameters, chosen so that the output becomes constant. This (UF) restriction decreases a by one, so the restricted image still contains an affine subspace of dimension at least $a - 1 \geq d$, a contradiction.

If the output is a literal of an AND-type gate on two unpaired parameters x and y , fix y to a constant such that the AND-gate becomes constant. Again, the output becomes constant while a decreases by only one, contradicting Proposition 2.2. $\square$

3 Deletion and reparameterisation

We now describe in more detail the deletion and reparametrisation operations used by the gate elimination algorithm, and record how each operation changes the quantities s , m , q , a , and the potential W .

3.1 Deletions

We characterize deletions as *gate deletions* (D1–D3) and *variable deletions* (D4). Each gate deletion removes a gate or an equation without changing the map Φ , whereas each variable deletion removes an equation that is the child of an unpaired, isolated parameter x , after exchanging x with an auxiliary variable.

D1: deleting a gate that depends on at most one input. Suppose that a gate J in the acyclic part computes a constant or a literal of one of its inputs. This is the case for every constant or unary gate, for every gate with a constant input, for every gate whose two inputs are identical, and for an AND-type gate whose two inputs are parameters or auxiliary variables with dependent linear parts (as affine functions of the parameters). We reroute every use of J to the constant or literal of its input, and delete J .

This deletion does not change the affine system, and every gate that was a child of J now reads a constant, or a literal of an input of J . We have $\Delta m = \Delta q = \Delta a = 0$ since the parameters and the pairs are unchanged, and $\Delta s = -1$, as gate J is deleted. An unpaired parameter that was a parent of J increases its weight from 6 to 7, if and only if, it was isolated (that is, its only child was J), and has at least two children after rerouting. Thus, $\Delta W \leq -2 - 6 + 7 = -1$, in this case. If J is rerouted to a constant, then no unpaired parameter can increase its weight, and so $\Delta W \leq -2$.

D2: deleting an unused gate. A gate that has no children and is not the designated output can be deleted without changing the computed function. In this case, $\Delta m = \Delta q = \Delta a = 0$, and $\Delta s = -1$ as one gate is deleted. Thus, $\Delta W \leq -2$.

D3: deleting an equation with at most two nonconstant terms. Let E be an equation in the affine system with at most two nonconstant terms. It must contain an auxiliary variable, otherwise the row corresponding to E in the auxiliary matrix would be zero, contradicting its invertibility. Therefore, we can write the equation E as $v = U + c$, where v is an auxiliary variable and U is either a parameter, an auxiliary variable, or $U = 0$. We substitute $v = U + c$ for all occurrences of v , and delete both v and the equation $v = U + c$ from the affine system.

As in (D1), an unpaired, isolated parameter can increase its weight from 6 to 7, if its only child was E and it has at least two children after the substitution. We have $\Delta m = \Delta q = \Delta a = 0$ as no parameters or pairs are removed, $\Delta s = -1$ as equation E is deleted, and thus, $\Delta W \leq -2 - 6 + 7 = -1$. If no unpaired parameter increases its weight after substitution, then the stronger bound $\Delta W \leq -2$ holds.

D4: unit parameter deletion. Consider an unpaired, isolated parameter x whose unique child is an equation E . We exchange x with a suitable auxiliary variable Q using the (AR) operation, and then delete x , which is now an auxiliary variable, together with equation E .

To find a suitable auxiliary variable Q for applying (AR) operation, write the affine system as $A\vec{v} = N\vec{u} + \vec{c}$. Since parameter x only occurs in equation E , the column N_x (corresponding to x) in the matrix N has a single non-zero entry in the row corresponding to E , so $A^{-1}N_x \neq 0$. Choose an auxiliary variable Q whose coordinate in $A^{-1}N_x$ is 1, in other words, we choose any auxiliary variable Q that depends on x (after expressing auxiliary variables as affine functions of the parameters). Then $Q = x + \varphi(\vec{u}_{-x})$, for some affine map φ . Now we apply the (AR) operation with this choice of auxiliary variable Q . After exchanging the roles of parameter x and auxiliary variable Q in the affine system, no equations are changed, so the new auxiliary variable x still occurs only in equation E .

The old parameter x of weight 6 is removed by the operation, and is replaced by the new parameter Q , whose weight is at most 7, and no other unpaired parameter increases its weight. As Q replaces x as a parameter and no pairs are changed, we have $\Delta m = \Delta q = \Delta a = 0$. Moreover, $\Delta s = -1$ because equation E is deleted, so $\Delta W \leq -2 - 6 + 7 = -1$.

3.2 Affine and quadratic reparameterisation

Recall from Section 2.5 that both reparameterisations (AR) and (QR) are bijective changes of parameters that preserve both the image and the form of map Φ . Then, Section 2.6 explains how they are implemented in the circuit so that every equation has at most three nonconstant terms and the auxiliary matrix remains invertible. We now describe the implementation of these reparametrisations in more detail, in the form used by the gate elimination algorithm, and record their effect on the quantities m , q , a , s , and the potential W .

AR: exchanging a parameter and an auxiliary variable. Let Q be an auxiliary variable which depends on an unpaired parameter x (as an affine function of the parameters), that is $Q = x + \varphi(\vec{u}_{-x})$ for some affine map φ . Note that Q and x may not occur in a common equation, but x must occur in some equation of the affine system, as otherwise the column corresponding to x in the parameter matrix N , and hence in $A^{-1}N$, would be zero, contradicting that Q depends on x . We substitute $x \leftarrow w + \varphi(\vec{u}_{-x})$ in the map Φ , regard Q as the new parameter w and x as an auxiliary variable. Note that this does not change any equation or gate of the circuit. In particular, every parameter other than x and w has the same set of children as before.

As the new parameter w replaces parameter x , x replaces auxiliary variable Q , and all pairs, equations, and gates of the circuit are unchanged, we obtain $\Delta m = \Delta q = \Delta a = \Delta s = 0$. Hence ΔW is the change in the total weight of the unpaired parameters, to which only x and w contribute, that is, $\Delta W = \text{wt}'(w) - \text{wt}(x)$. The new parameter w has exactly the same set of children as Q , so $\deg'(w) = \deg(Q)$. Moreover, w occurs in at least one equation, since the column corresponding to Q in the invertible matrix A is nonzero. If auxiliary variable Q has at least one other child in the old circuit, then $\deg'(w) \geq 2$. If, in addition, $\deg(x) \geq 2$ in the old circuit, then both weights equal 7, and so $\Delta W = 0$. We apply (AR) in only two situations: in (D4), where x is isolated and the possible increase of one is offset by the deletion of an equation, and in Section 4.2, where $\deg(x) \geq 2$ and Q is an input of the minimal AND, so that $\deg'(w) \geq 2$ and $\Delta W = 0$.

Unlike an affine substitution in the setting of [19], which always reduces the number of free variables, (AR) keeps the number of parameters, the circuit, and $\text{im } \Phi$ unchanged. Therefore, we can apply (AR) whenever it brings the circuit into a more convenient form, for instance to turn both inputs of a minimal AND into parameters before applying a restriction, as we do in Section 4.2. This extra freedom is one of the main sources of our improved lower bound.

QR: a quadratic change of parameters. We apply (QR) only to a specific configuration, which we describe below. Let x, y, z be distinct unpaired parameters, and suppose that, for some constants $\alpha, \beta, \gamma, c \in \mathbb{F}_2$, the circuit contains the AND-type gate $G = (x + \alpha)(y + \beta) + \gamma$, and XOR-type gate $D = G + z + c$, where z is a isolated parameter whose only child is D , and the only child of gate G is D . We perform the following quadratic substitution on the parameter z ,

$$z \leftarrow w + (x + \alpha)(y + \beta) + \gamma + c$$

in the map Φ , and add the pair $\{x, y\}$ to P . Under this substitution, $D = G + (w + G + c) + c = w$, so we reroute every use of D to the new parameter w , and delete D . Since the unpaired,

isolated parameter z had only gate D as its child, the (QR) substitution does not change the rest of the circuit. The gate G becomes unused after substitution, so we delete it using (D2).

Since new parameter w replaces parameter z , one pair $\{x, y\}$ is added and two gates are deleted, we have $\Delta m = 0$, $\Delta q = 1$, $\Delta a = -1$, and $\Delta s = -2$. For the potential W , the two gate deletions contribute -4 , the new pair $\{x, y\}$ contributes $+3$, while the parameters x and y lose their weights since they are now paired. The new parameter w , whose children are exactly those of D in the previous circuit, has weight at most 7 and replaces unpaired, isolated parameter z of weight 6. No other unpaired parameter of the system gets a new child after applying (QR). So, we have

$$\Delta W \leq -4 + 3 + 7 - 6 - \text{wt}(x) - \text{wt}(y) = -\text{wt}(x) - \text{wt}(y).$$

We apply (QR) only when $\deg(x), \deg(y) \geq 2$, so both weights equal 7 and $\Delta W \leq -14$.

4 The gate elimination algorithm

Theorem 4.1. *For every affine disperser $f : \{0, 1\}^n \rightarrow \{0, 1\}$ for dimension d , any B_2 -circuit C computing f has size at least $3.5n - 7d$.*

The proof of Theorem 4.1 is via a gate elimination algorithm, and is given at the end of this section. We obtain Theorem 1.1 as a corollary by using Li's result [37] on an explicit affine disperser for dimension $O(\log(n))$.

Most of this section will be used to describe the gate elimination algorithm. The algorithm runs until $a \leq d$, and it terminates because each step decreases $m + s$. The key property of the algorithm is captured in the following lemma, which states that as long as $a > d$, we can decrease the potential W by at least 14 per unit of decrease in quantity a .

Lemma 4.2. *Suppose that $a > d$. Then, in polynomial time, the gate elimination algorithm transforms the current state into a new state such that $m+s$ strictly decreases, $\Delta a \in \{0, -1, -2\}$, and $\Delta W \leq 14\Delta a$.*

Each iteration performs the first applicable reduction from the cases below, possibly after several preparatory operations (moving affine gates into the affine system, affine reparameterizations, and reroutings) that leave m , s , and a unchanged, and do not increase W .

4.1 Preliminary reductions

First, we move every gate with an affine operation and no AND-type ancestor into the affine system, as described in Section 2.3. Each such move replaces one gate by one equation, and does not increase the degree of any parameter, so it satisfies $\Delta m = \Delta q = \Delta a = \Delta s = 0$ and thus, $\Delta W \leq 0$. We then check the following six cases in order, and perform the reduction of the first case that is applicable. Therefore, for each case that applies, we may assume that all earlier cases are inapplicable.

Case 1: a deletion is available. We repeatedly apply all of the deletions from (D1)–(D4), in order, until none of them are applicable. Recall that each deletion satisfies $\Delta m = \Delta q = \Delta a = 0$, $\Delta s = -1$, and $\Delta W \leq -1$.

If no deletion applies, then the circuit has the following properties. Since (D1) does not apply, every gate is XOR-type or AND-type and has two distinct inputs, neither of which is a constant. Moreover, if both inputs of an AND-type gate are parameters or auxiliary variables,

then they must have independent linear parts. Since (D2) and (D3) do not apply, every gate other than the output has a child, and every equation in the affine system has exactly three nonconstant terms. Finally, since (D4) does not apply, no unpaired isolated parameter occurs in the affine system.

Case 2: an AND-type gate on parameters has an unpaired, isolated input. Let G be an AND-type gate whose inputs are parameters z and w , where z is an unpaired, isolated parameter. We fix w to a constant such that the AND-type gate G becomes constant, delete gate G using deletion (D1), and reroute all occurrences of G to that constant. As G was the only child of z , the parameter z is now unused in the circuit. Note that the degree of all other parameters in the circuit is unchanged, as G is rerouted to a constant.

If parameter w is unpaired, we use the (UF) restriction to fix w to a constant. Note that both unpaired parameters z and w had weights at least 6 before they are removed. Thus, we have $\Delta m = -1$, $\Delta s = -1$, $\Delta q = 0$, $\Delta a = -1$, and so $\Delta W \leq -6 - 6 - 2 = -14$.

If w is a paired parameter, we use the (PF) restriction to fix w to a constant. Let r be the partner of w . After the (PF) restriction, the pair (r, w) is removed, and r becomes an unpaired parameter with weight at most 7. Consequently, we have $\Delta m = -1$ (as w is fixed to a constant), $\Delta s = -1$ (as gate G is deleted), $\Delta q = -1$ (as pair (r, w) is removed), $\Delta a = \Delta m - \Delta q = 0$. For the potential W , removing the pair (r, w) contributes -3 , losing the weight of z contributes -6 , deleting G contributes -2 , and the weight of parameter r (after it is unpaired) contributes at most $+7$. Thus, we have $\Delta W \leq -3 - 6 - 2 + 7 = -4$.

We now record the consequence of inapplicability of Case 1 and Case 2. For every isolated, unpaired parameter z , it is not the output (by Lemma 2.3), and does not occur in the affine system (by inapplicability of Case 1). So, the unique child of z must be a gate G in the acyclic part of the circuit. The other input of gate G cannot be a parameter. We show this by taking cases on gate G being XOR-type or AND-type. If G is a XOR-type gate with both of its inputs being parameters, it would already have been moved into the affine system. If G is an AND-type gate, this would make Case 2 applicable. Thus, no gate or equation in the circuit has both an isolated unpaired parameter and a different parameter as its parents.

Case 3: a paired parameter has degree zero or at least two. Let p be a paired parameter with partner r . If $\deg(p) = 0$, we use (PF) to fix r to a constant. The pair (p, r) is removed, and p becomes an unpaired parameter, which does not contribute to the potential W as $\deg(p) = 0$. Consequently, we have $\Delta m = -1$ (as r is fixed to a constant), $\Delta s = 0$ (as no gates are deleted), $\Delta q = -1$ (as pair (p, r) is removed), $\Delta a = \Delta m - \Delta q = 0$. This gives, $\Delta W \leq -3$.

If $\deg(p) \geq 2$, we fix p to a constant using (PF), and delete two of its children, using the deletions (D1)–(D4). Observe that these two deletions of p 's children does not increase the weight of any parameter which was unpaired initially. Only the parameter r , which becomes newly unpaired after the pair (p, r) is removed, can increase its weight to at most 7. Note that $\Delta m = -1$ (as p is fixed to a constant), $\Delta s = -2$ (as two children of p are deleted), $\Delta q = -1$ (as the pair (p, r) is removed), and $\Delta a = \Delta m - \Delta q = 0$. Thus, we have $\Delta W \leq -3 - 4 + 7 = 0$.

If Case 3 is inapplicable, then every paired parameter is isolated, and has exactly one child.

Case 4: two partners have the same child. Let p be a paired parameter with partner r such that p and r have a common child H . If H is an equation, we can write it as $v + p + r = c$ for an auxiliary variable v . If H is a gate, its two inputs are p and r . Use the (PF) restriction to impose $p = r + c'$, choosing the constant c' such that H becomes constant. Any choice

of c' works if H is an equation or a XOR-type gate. If H is an AND-type gate of the form $(p + \alpha)(r + \beta) + \gamma$ (where $\alpha, \beta, \gamma \in \mathbb{F}_2$ are constants), we choose $c' = \alpha + \beta + 1$.

After H becomes constant, we delete H , and reroute the occurrences of H to its constant value. The parameter r survives, but is now unused, since H was its only child. Note that $\Delta m = -1$ (as p is restricted), $\Delta s = -1$ (as H is deleted), $\Delta q = -1$ (as pair (p, r) is removed), and $\Delta a = \Delta m - \Delta q = 0$. Thus, we have $\Delta W \leq -3 - 2 = -5$.

Case 5: a paired parameter occurs in the affine system. Let p be a paired parameter that occurs in the affine system, and let r be its partner. Since Case 3 does not apply, the paired parameter p must have a unique child H_p , which must be an equation since p occurs in the affine system. Similarly, r has a unique child H_r , and $H_r \neq H_p$ since Case 4 does not apply.

We fix parameter r to a constant using (PF) restriction and delete H_r , which is now deletable by (D1) if it is a gate and by (D3) if it is an equation. Since parameter r is a parent of H_r , and both Case 1 and Case 2 do not apply, H_r does not have any isolated unpaired parameter as its parent. Therefore, deleting H_r does not increase the weight of any parameter which was unpaired initially. After r is fixed to a constant, p becomes an unpaired parameter whose unique child is H_p , so it gets a weight of 6. So far, removing the pair (p, r) contributes -3 , deleting H_r contributes -2 , and the new weight of p contributes 6, so the potential W has increased by at most $\Delta W \leq -3 - 2 + 6 = 1$.

Now, we apply (D4) to parameter p and equation H_p , this is applicable as p is an unpaired, isolated parameter whose only child is equation H_p . As explained in Section 3.1, this decreases the potential W by at least one, that is, $\Delta W \leq -1$. Altogether, we have $\Delta m = -1$ (as parameter r is fixed to a constant), $\Delta s = -2$ (as H_r and H_p are deleted), $\Delta q = -1$ (as pair (p, r) is removed), and $\Delta a = \Delta m - \Delta q = 0$. Finally, we have $\Delta W \leq +1 - 1 = 0$.

Case 6: an AND-type gate on parameters has a paired input. Let G be an AND-type gate whose inputs are parameters p and w , where p is a paired parameter with partner r . As Case 3 does not apply, G must be the unique child of paired parameter p . Moreover, the paired parameter r has a unique child H_r . Since Case 5 does not apply, r does not occur in the affine system, so H_r is a gate. Since Case 4 does not apply, we must have $H_r \neq G$, and $w \neq r$. Further, $w \neq p$, as Case 1 does not apply. Since Case 2 does not apply, we must have that w is not an unpaired isolated parameter. Therefore, w is either an unpaired parameter of degree at least two, or a paired parameter.

We fix r to a constant using (PF) restriction, and then fix parameter w to a constant so that gate G becomes constant, this is by a (UF) restriction if w is unpaired and by a (PF) restriction if w is paired. Then we delete the gates G and H_r using (D1). After r is restricted, the parameter p becomes unpaired and unused, as it was isolated initially (the only child of p was gate G). Observe that the weight of any parameter that was initially unpaired cannot increase: G is rerouted to a constant after deletion, and since r is a parent of H_r and both Case 1 and Case 2 do not apply, H_r has no unpaired isolated parameter as a parent.

If w was an unpaired parameter initially, its degree was at least 2, and its weight was contributing $+7$ to the potential W . After w is restricted, it loses its contribution of $+7$ to W . In this case, we have $\Delta m = -2$ (as parameters r and w are restricted), $\Delta s = -2$ (as G and H_r are deleted), $\Delta q = -1$ (as pair (p, r) is removed), and $\Delta a = \Delta m - \Delta q = -1$. This gives $\Delta W \leq -7 - 3 - 4 = -14$.

If w was a paired parameter initially (with partner r'), two pairs (p, r) and (w, r') are removed after the restriction of r and w . The parameter p becomes unpaired and unused as its only child was G . Thus, only parameter r' is the newly unpaired parameter that can have

positive weight, and its weight is at most 7. In this case, we have $\Delta m = -2$ (as two parameters r and w are restricted), $\Delta s = -2$ (as G and H_r are deleted), $\Delta q = -2$ (as two pairs (p, r) and (w, r') are removed), and $\Delta a = \Delta m - \Delta q = 0$. Thus, we have, $\Delta W \leq -6 - 4 + 7 = -3$.

If none of the Cases 1–6 applies to a state, then every parameter that occurs in the affine system is unpaired and has degree at least two, and for every AND-type gate G with parameters x and y as its inputs, both x and y must be unpaired parameters, and $\deg(x), \deg(y) \geq 2$.

4.2 Restructuring the minimal AND

Suppose that $a > d$, and that none of the preliminary reductions applies to the current state. Then, the acyclic part has at least one gate, otherwise the output would be an affine function of the parameters, contradicting Lemma 2.3. We choose the first gate G in the acyclic part according to the topological order. Observe that G must be an AND-type gate, as all the affine gates, both of whose inputs are in the affine system, have already been moved into the affine system at the start of the preliminary reductions. We call the gate G a *minimal AND*. Moreover, as Case 1 does not apply, both inputs of G are parameters or auxiliary variables, and their linear parts, as affine functions of the parameters, are independent.

Transforming the inputs to parameters. Suppose that one input of G is an auxiliary variable Q . As the affine system has a unique solution, the auxiliary variable Q is an affine function of the parameters, given by the row corresponding to Q in $\vec{v} = A^{-1}N\vec{u} + A^{-1}\vec{c}$. As the inputs of gate G have independent linear parts, the auxiliary variable Q is nonconstant, and depends on some parameter x . The parameter x must occur in some equation of the affine system. To see this, observe that if x did not occur in any equation, the column corresponding to x in the parameter matrix N would be zero, which implies that the column corresponding to x in $A^{-1}N$ would be zero. This would contradict the dependence of Q on x .

Since no case of the preliminary reductions applies to the current state, every parameter that occurs in the affine system is unpaired and has degree at least 2. In particular, x is an unpaired parameter, and $\deg(x) \geq 2$.

We apply (AR) to exchange Q and x , as explained in Section 3.2. The new parameter w that replaces auxiliary variable Q has degree at least two, because Q occurs in an equation (the column corresponding to Q in the invertible auxiliary matrix is nonzero), and is also a parent of gate G . As $\deg(x) \geq 2$, both x and w have weight 7, so we have $\Delta m = \Delta s = \Delta q = \Delta a = \Delta W = 0$.

We move all affine gates into the affine system again if applicable, and then inspect all the cases in the preliminary reductions, in order. If one of the Cases 1–6 applies, we perform it, and end the current iteration of restructuring the minimal AND. If no preliminary reduction applies, we proceed with the current iteration of restructuring the minimal AND G .

If the other input of G is also an auxiliary variable, we repeat the above step again, to exchange the auxiliary variable with a parameter on which it depends (other than w). Such a parameter exists because the two inputs of G were linearly independent before the application of (AR), and an invertible affine change of parameters preserves the linear independence of the two inputs of G . Thus, after at most two applications of (AR), both inputs of G are parameters, which we denote by x and y from now on. Since Case 2 and Case 6 do not apply, x and y are unpaired parameters, and $\deg(x), \deg(y) \geq 2$.

Rerouting to ensure that no child of G reads x or y . Suppose that a child J of minimal AND G also has parameter x as a parent. Note that J must be a gate in the acyclic part

of the circuit, as there are no wires going back from the acyclic part to the affine system. The value computed at gate J has the form $h(x, G(x, y))$, where $h \in B_2$ depends only on the two parameters x and y . We relabel J by the operation $(x, y) \mapsto h(x, G(x, y))$, and reroute its inputs to x and y . Observe that the function computed by the circuit is not changed by this rerouting. After the rerouting, J is no longer a child of G but is still a child of x , and y is the only parameter that can obtain new occurrences. Since $\deg(y) \geq 2$ initially, the weight of y after rerouting still remains 7. Moreover, the same operation can be applied with the roles of x and y interchanged. As no parameters, gates or pairs are removed, we have $\Delta m = \Delta s = \Delta q = \Delta a = 0$, and as no unpaired parameter increases its weight, $\Delta W \leq 0$.

After each rerouting, we move affine gates into the system, and inspect the cases of the preliminary reductions in order, ending the current iteration if one case applies. Each rerouting removes a child of G , and none of the preparatory operations creates a new child of G , so the rerouting process terminates after finitely many steps. If the rerouting ends in a state where no preliminary reduction is applicable, then no child of G has x or y as an input. By Lemma 2.3, the gate G is not the output, and has at least one child, as Case 1 does not apply.

4.3 Eliminating the minimal AND

We may assume that $\deg(x) \geq \deg(y) \geq 2$ by swapping x and y if necessary, and let $k \geq 1$ be the number of children of G . We first work with the case in which both parameters x and y have degree 2, and share a common second child.

Case 7: $\deg(x) = \deg(y) = 2$ and there is a common second child for x and y . Suppose that $\deg(x) = \deg(y) = 2$ and that x and y have the same second child $B \neq G$. If B is an equation, then it has exactly three nonconstant terms, since Case 1 does not apply, and at least one of the terms is an auxiliary variable (as the auxiliary matrix is invertible). Hence, B is of the form $v + x + y = c$, where v is an auxiliary variable, and $c \in \mathbb{F}_2$ is a constant. If B is not an equation, it must be a gate in the acyclic part with inputs x and y . Set $h(x, y) = x + y$ (if B is an equation), or set h to be the operation of B (if B is a gate).

As a function of x and y , the AND-type gate G takes the same value at three points of $\mathbb{F}_2^2$, and at least two of these points have the same value under h (as the range of h is $\mathbb{F}_2$). These two points form a line of the form $\alpha x + \beta y = \gamma$ in $\mathbb{F}_2^2$ (where $\alpha, \beta, \gamma \in \mathbb{F}_2$ are constants, and $(\alpha, \beta) \neq (0, 0)$) such that both G and h are constant on this line. We restrict to this line by a single application of (UF): we fix $x = \gamma$ if $\beta = 0$, or fix $y = \gamma$ if $\alpha = 0$, or impose $x = y + \gamma$ if $\alpha = \beta = 1$. After restricting to the line, G computes a constant, and so does B if it is a gate. If B is an equation, it reduces to the form $v = c_0$ (for some constant $c_0 \in \mathbb{F}_2$). We delete G by (D1) and B by (D1) or (D3), rerouting G and B (or v , if B is an equation) to the respective constant value they compute. The restriction removes one of x and y , and the other becomes unused, since all occurrences of x and y were in G and B . Thus, x and y both lose their weight contribution of 7 to W , and no other unpaired parameter increases its weight, as we rerouted only to constants. We have $\Delta m = -1$ (as (UF) is applied once), $\Delta s = -2$ (as G and B are deleted), $\Delta q = 0$ (as no pairs are removed), and $\Delta a = \Delta m - \Delta q = -1$. Hence, $\Delta W \leq -7 - 7 - 4 = -18$.

From now on, we may assume that this case is not applicable for x and y . In particular, if $\deg(x) = \deg(y) = 2$, then the second children of x and y (which we denote by B_x and B_y , respectively) are distinct.

Case 8: $\deg(x) \geq 3$, or G has at least two children, or its unique child has no unpaired, isolated input. Let k_1 be the number of children of G whose other parent is

an unpaired isolated parameter. Observe that we have $0 \leq k_1 \leq k$. We fix parameter x to a constant such that G becomes constant, this is a (UF) restriction (as x is unpaired). We then delete all the $\deg(x)$ children of x , starting with G , and then, all the k children of G . These $\deg(x) + k$ internal values are distinct, as no child of G has x as a parent. Once parameter x is fixed, every child of x is deletable, by (D1) if it is a gate and by (D3) if it is an equation. Once G is rerouted to its constant value, every child of G is a gate of the acyclic part, and is deletable by (D1).

As the unpaired parameter x initially has $\deg(x) \geq 2$, it loses its initial weight of 7 after restriction. Each of the $\deg(x) + k$ deletions decreases the potential W by -2 . Now we bound the increase in weights of other unpaired parameters. An unpaired parameter whose degree is at least two already has the maximum weight 7, so its weight cannot increase after rerouting. For every unpaired parameter which was initially unused, it still remains unused after rerouting. For every unpaired parameter z which was isolated initially, it can increase its weight after rerouting only if the unique child of z is deleted, and rerouted to a literal of z . Since Case 1 and Case 2 do not apply, no gate or equation has both an unpaired, isolated parameter and another parameter among its parents, so the unique child of z is not a child of x . Therefore, the unique child of z must be a child of G . That is, the child of z must be one of the k_1 children of G whose other parent is an unpaired, isolated parameter. Hence, at most k_1 unpaired parameters can increase their weight after rerouting, each by at most one, from 6 to 7. Therefore, $\Delta m = -1$, $\Delta s = -(\deg(x) + k)$, $\Delta q = 0$, $\Delta a = -1$, and

$$\Delta W \leq -7 - 2(\deg(x) + k) + k_1.$$

If $\deg(x) = 2$, then we must also have $\deg(y) = 2$, as $\deg(x) \geq \deg(y) \geq 2$. There is an additional decrease in the potential W in this case, by a careful accounting of the weight of unpaired parameter y .

The children of y are G and B_y , in particular, y is not a parent of B_x , since $B_x \neq B_y$. Moreover, no child of G has y as a parent. As G is rerouted to a constant, y does not obtain a new child during this step, and it loses the child G , so it becomes an isolated parameter. Hence, the weight of y drops from 7 to at most 6, after rerouting G to a constant.

Combining all the estimates gives,

$$\Delta W \leq \begin{cases} -7 - 2\deg(x) - k \leq -14, & \deg(x) \geq 3, \\ -12 - 2k + k_1 \leq -14, & \deg(x) = 2, k \geq 2, \\ -14 + k_1, & \deg(x) = 2, k = 1. \end{cases}$$

Since $\Delta a = -1$, Case 8 satisfies $\Delta W \leq 14\Delta a$, as required by Lemma 4.2, unless $\deg(x) = \deg(y) = 2$ and $k = k_1 = 1$.

We call the case where $\deg(x) = \deg(y) = 2$ and $k = k_1 = 1$ the *bottleneck configuration*: here G has a unique child D , whose other parent is an unpaired, isolated parameter z , see Figure 3 for an illustration. Since Case 1 does not apply, D is either AND-type or XOR-type. We check for this configuration in the circuit before applying Case 8, and, if it occurs, apply the corresponding one of the following two cases instead.

Case 9: The bottleneck configuration where D is AND-type. Since D is AND-type, we can write it in the form

$$D = (G + \delta)(z + \eta) + \theta$$

for some constants $\delta, \eta, \theta \in \mathbb{F}_2$. Since G is nonconstant, we can choose $x_0, y_0 \in \mathbb{F}_2$ with $G(x_0, y_0) = \delta$. We fix $x = x_0$ and $y = y_0$, which are two (UF) restrictions, as x and y are

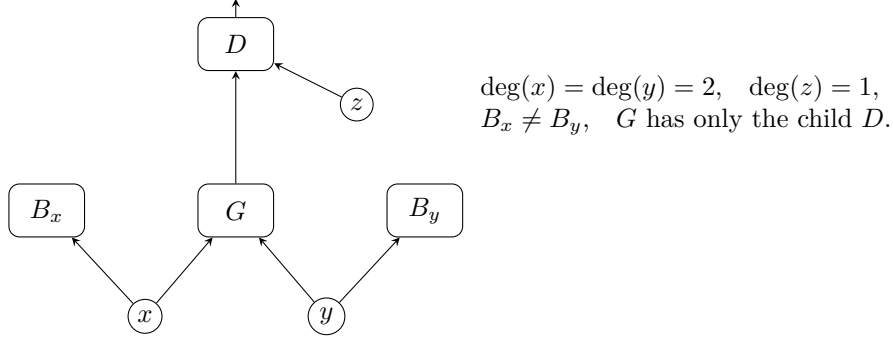


Figure 3: The bottleneck configuration.

unpaired. Then G computes the constant δ , and hence D computes the constant θ . We delete G and D by (D1), rerouting them to their respective constant values. We also delete B_x and B_y , each by (D1) if it is a gate and by (D3) if it is an equation, which is possible since x and y have been fixed. These four gates and equations are distinct: B_x and B_y are distinct second children of x and y , and neither of them equals D , since no child of G has x or y as a parent.

The only child of z was D , which is rerouted to a constant, so z becomes unused. As in the general restriction, an unpaired parameter can increase its weight only if it is initially isolated and its unique child is deleted and rerouted to a literal of it. This does not happen here: G and D are rerouted to constants, and neither B_x nor B_y has an unpaired, isolated parameter as a parent, since x is a parent of B_x , y is a parent of B_y , and both Case 1, Case 2 do not apply. Thus x , y , and z lose their weights 7, 7, and 6, and the four deletions contribute -8 to ΔW . Altogether, $\Delta m = -2$, $\Delta s = -4$, $\Delta q = 0$, $\Delta a = -2$, and

$$\Delta W \leq -7 - 7 - 6 - 8 = -28 = 14\Delta a.$$

Case 10: The bottleneck configuration where D is XOR-type. Since D is XOR-type, we can write $D = G + z + c$ for some constant $c \in \mathbb{F}_2$. Recall that this is precisely the configuration in which we apply (QR): x , y , and z are distinct unpaired parameters, G is an AND-type gate on x and y whose only child is D , and z is isolated with the only child D . We apply (QR) with the substitution

$$z \leftarrow w + G(x, y) + c,$$

where w is a fresh parameter and $G(x, y)$ is the quadratic polynomial computed by the AND-type gate G . This change of parameters is bijective, it adds the pair $\{x, y\}$ to P , and gate D now computes $G + (w + G + c) + c = w$. We reroute every use of D to w , and delete D . Then G has no child and is not the output of the circuit, so we delete it using (D2). As the only child of z was gate D before substitution, no equation of the affine system, and no other gate of the acyclic part is affected by this (QR) application.

As two gates G and D are deleted, this contributes -4 to potential W . The parameters x and y become paired, so each of them lose their individual weight of 7, while the new pair (x, y) contributes $+3$ to W . The new parameter w replaces z , and the children of w are exactly the former children of D . So the weight z , which was 6 before substitution, is replaced by the weight of new parameter w , which is at most 7. Note that no other unpaired parameter increases its weight, as D is rerouted to fresh parameter w , and G is deleted without rerouting. Altogether, $\Delta m = 0$, $\Delta s = -2$, $\Delta q = 1$, $\Delta a = -1$, and

$$\Delta W \leq -4 - 7 - 7 - 6 + 3 + 7 = -14 = 14\Delta a.$$

Proof of Lemma 4.2. Since $a > d$, the case analysis above shows that each iteration performs one of the reductions, possibly after some preparatory operations. The preparatory operations, which are moves of affine gates into the affine system, affine reparametrisations, and reroutings, preserve m , s , and a , and do not increase W . We have shown in each case that the reduction satisfies $\Delta a \in \{0, -1, -2\}$ and $\Delta W \leq 14\Delta a$, so the same holds for the whole iteration. Moreover, the deletions (D1)–(D4) and (QR) decrease s and leave m unchanged, every other reduction decreases m , and no reduction increases s . Therefore, each iteration strictly decreases $m + s$.

An iteration of the procedure above performs at most s moves of affine gates into the affine system, at most two affine exchanges, and at most one rerouting per child of the minimal AND. We use Gaussian elimination over $\mathbb{F}_2$ to express the auxiliary variables as affine functions of the parameters, which allows us to test linear dependence and to perform (AR). All remaining tests and updates only involve the lists of gates and equations and Boolean truth tables of constant size. Hence each iteration runs in polynomial time, which completes the proof. $\square$

Now, we complete the proof of Theorem 4.1, and observe that no reweighting of the potential function can improve the leading coefficient of 3.5 in our analysis.

Proof of Theorem 4.1. We initialise the state with $\Phi(\vec{u}) = \vec{u}$, no pairs, an empty affine system, and C as the acyclic part. Then $m = a = n$, $q = 0$, and $s = |C|$. Since $q = 0$ and every input is an unpaired parameter of weight at most 7, the initial potential satisfies $W_0 \leq 2|C| + 7n$.

While $a > d$, we apply Lemma 4.2: we first try the preliminary reductions and, if none of them applies, follow the restructuring and elimination procedures of Sections 4.2 and 4.3. By Lemma 4.2, each iteration strictly decreases $m + s$. Since $m + s = n + |C|$ initially and $m + s \geq 0$ throughout, the algorithm stops with $a \leq d$ after at most $n + |C|$ iterations.

We store Φ through the vector $\vec{b}$, the matrix L , the set P , and the vectors $\vec{c}_{p,r}$ in (1), all of which can be updated in polynomial time after each operation. Together with Lemma 4.2, and since $m \leq n$ and $s \leq |C|$ throughout, each iteration takes time polynomial in n and $|C|$, so the algorithm runs in polynomial time.

Let T denote the number of iterations of the algorithm, so that $a_T \leq d$. Summing the bound $\Delta W \leq 14\Delta a$ of Lemma 4.2 over all iterations gives

$$W_T - W_0 \leq 14(a_T - n).$$

Since $W_T \geq 0$ and $a_T \leq d$, it follows that

$$2|C| + 7n \geq W_0 \geq W_T + 14(n - a_T) \geq 14(n - d).$$

Rearranging gives

$$|C| \geq 3.5n - 7d,$$

which proves the theorem. $\square$

Theorem 1.1 follows from Theorem 4.1 by using the result of Li [37] that there is an affine disperser $f \in \mathcal{P}$ for dimension $O(\log(n))$.

Remark 4.3 (Why this analysis gives 7/2). Consider any potential of the form

$$V = s + \beta q + \alpha I - \sigma \ell,$$

where I is the number of used unpaired parameters, ℓ is the number of unpaired, isolated parameters, and the weights satisfy $\alpha \geq \sigma \geq 0$ and $\beta \geq 0$. A tedious but straightforward check of our estimates shows that the weights chosen in our proof, $\alpha = 7/2$, $\beta = 3/2$, and $\sigma = 1/2$, are optimal, that is, no other choice of weights can improve the coefficient 7/2.

5 Acknowledgment

We thank Mahyar Afshinmehr for collaboration at an early stage of the project.

The authors would like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the programme Logical Foundations of Complexity Theory (LFC), during which this work was undertaken.

This project has received funding from the European Research Council under the European Union’s Horizon 2020 Research and Innovation Programme (Grant Agreement No. 101201339).

References

- [1] Miklós Ajtai. Σ^1_1 -formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983.
- [2] Kazuyuki Amano and Jun Tarui. A well-mixed function with circuit complexity $5n$: Tightness of the lachish-raz-type bounds. *Theor. Comput. Sci.*, 412(18):1646–1651, 2011.
- [3] Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the $P = ? NP$ question. *SIAM J. Comput.*, 4(4):431–442, 1975.
- [4] Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov, and Avi Wigderson. Simulating independence: New constructions of condensers, ramsey graphs, dispersers, and extractors. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 1–10, 2005.
- [5] Eli Ben-Sasson and Swastik Kopparty. Affine dispersers from subspace polynomials. *SIAM J. Comput.*, 41(4):880–914, 2012.
- [6] Eli Ben-Sasson and Emanuele Viola. Short pcps with projection queries. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 163–173. Springer, 2014.
- [7] Norbert Blum. A boolean function requiring $3n$ network size. *Theor. Comput. Sci.*, 28:337–345, 1984.
- [8] B.M.Kloss and V.A.Malyshchev. Estimates of the complexity of certain classes of functions. *Vestn. Moskov. Univ. Ser.*, 1(4):44–51, 1965.
- [9] Ravi B. Boppana and Michael Sipser. The complexity of finite functions. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 757–804. Elsevier and MIT Press, 1990.
- [10] Jean Bourgain. On the construction of affine extractors. *GAFA Geometric And Functional Analysis*, 17(1):33–57, 2007.
- [11] Jin-yi Cai. S_2^P is subset of zpp^{NP} . *J. Comput. Syst. Sci.*, 73(1):25–35, 2007.
- [12] Venkatesan T. Chakaravarthy and Sambuddha Roy. Oblivious symmetric alternation. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2006.

- [13] Eshan Chattopadhyay, Jesse Goodman, and Jyun-Jie Liao. Affine extractors for almost logarithmic entropy. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 622–633. IEEE, 2021.
- [14] Eshan Chattopadhyay and Jyun-Jie Liao. Extractors for sum of two sources. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1584–1597, 2022.
- [15] Ruiwen Chen and Valentine Kabanets. Correlation bounds and #sat algorithms for small linear-size circuits. *Theor. Comput. Sci.*, 654:2–10, 2016.
- [16] Evgeny Demenkov, Arist Kojevnikov, Alexander S. Kulikov, and Grigory Yaroslavtsev. New upper bounds on the boolean circuit complexity of symmetric functions. *Inf. Process. Lett.*, 110(7):264–267, 2010.
- [17] Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 2011.
- [18] Paul E. Dunne. *The Complexity of Boolean Networks*. Academic Press, 1988.
- [19] Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than- $3n$ lower bound for the circuit complexity of an explicit function. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, New Jersey, USA, October 9-11, 2016*, pages 89–98. IEEE Computer Society, 2016.
- [20] Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. Improving $3n$ circuit complexity lower bounds. *Comput. Complex.*, 32(2):13, 2023.
- [21] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009.
- [22] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984.
- [23] Alexander Golovnev. *Circuit Complexity: New Techniques and Their Limitations*. PhD thesis, New York University, USA, 2017.
- [24] Alexander Golovnev, Edward A. Hirsch, Alexander Knop, and Alexander S. Kulikov. On the limits of gate elimination. *J. Comput. Syst. Sci.*, 96:107–119, 2018.
- [25] Alexander Golovnev, Alexander S. Kulikov, Alexander V. Smal, and Suguru Tamaki. Circuit size lower bounds and #sat upper bounds through a general framework. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, Kraków, Poland, August 22-26, 2016*, volume 58 of *LIPICs*, pages 45:1–45:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [26] Alexander Golovnev, Alexander S. Kulikov, and R. Ryan Williams. Circuit depth reductions. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, Virtual Conference, January 6-8, 2021*, volume 185 of *LIPICs*, pages 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

- [27] Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986.
- [28] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.
- [29] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of $5n - o(n)$ for boolean circuits. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2002.
- [30] Hamidreza Jahanjou, Eric Miles, and Emanuele Viola. Local reduction. *Inf. Comput.*, 261:281–295, 2018.
- [31] Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.
- [32] Alexander S. Kulikov. Lower bounds for unrestricted boolean circuits: Open problems. In Fedor V. Fomin and Vladimir V. Podolskii, editors, *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, Proceedings*, volume 10846 of *Lecture Notes in Computer Science*, pages 15–22. Springer, 2018.
- [33] Oded Lachish and Ran Raz. Explicit lower bound of $4.5n - o(n)$ for boolean circuits. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 399–408. ACM, 2001.
- [34] Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1180–1193. ACM, 2022.
- [35] Xin Li. A new approach to affine extractors and dispersers. In *2011 IEEE 26th Annual Conference on Computational Complexity*, pages 137–147. IEEE, 2011.
- [36] Xin Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, New Jersey, USA, October 9-11, 2016*, pages 168–177. IEEE Computer Society, 2016.
- [37] Xin Li. Two source extractors for asymptotically optimal entropy, and (many) more. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1271–1281. IEEE, 2023.
- [38] Eric Miles and Emanuele Viola. Substitution-permutation networks, pseudorandom functions, and natural proofs. *J. ACM*, 62(6):46:1–46:29, 2015.
- [39] Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM J. Comput.*, 49(5), 2020.
- [40] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

- [41] Wolfgang J. Paul. A $2.5n$ -lower bound on the combinational complexity of boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977.
- [42] Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Matematicheskie Zametki*, 41(4):598–607, April 1987. In Russian. English Translation in [43].
- [43] Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.
- [44] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.
- [45] Rahul Santhanam. Circuit lower bounds for merlin–arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.
- [46] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, Nevada, USA, October 23-26, 2010*, pages 183–192. IEEE Computer Society, 2010.
- [47] John E. Savage. Computational work and time on finite machines. *J. ACM*, 19(4):660–674, 1972.
- [48] Claus-Peter Schnorr. Zwei lineare untere schranken für die komplexität boolescher fionen. *Computing*, 13(2):155–171, 1974.
- [49] Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Comput. Complex.*, 22(2):245–274, 2013.
- [50] Ronen Shaltiel. Dispersers for affine sources with sub-polynomial entropy. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 247–256. IEEE, 2011.
- [51] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Syst. Tech. J.*, 28(1):59–98, 1949.
- [52] Michael Sipser. Borel sets and circuit complexity. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 61–69. ACM, 1983.
- [53] Michael Sipser. The history and status of the P versus NP question. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 603–618. ACM, 1992.
- [54] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82. ACM, 1987.
- [55] Larry J. Stockmeyer. On the combinational complexity of certain symmetric boolean functions. *Math. Syst. Theory*, 10:323–336, 1977.

- [56] Ingo Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.
- [57] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.
- [58] Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.
- [59] Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 1–10. IEEE Computer Society, 1985.
- [60] Amir Yehudayoff. Affine extractors over prime fields. *Combinatorica*, 31(2):245–256, 2011.
- [61] Uri Zwick. A $4n$ lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic boolean functions. *SIAM J. Comput.*, 20(3):499–505, 1991.