Approximating the noise sensitivity of a monotone Boolean function

by

Arsen Vasilyan

B.S., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 15, 2020

Certified by . . . . . .
Ronitt Rubinfeld
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Approximating the noise sensitivity of a monotone Boolean function

by

Arsen Vasilyan

Submitted to the Department of Electrical Engineering and Computer Science
on May 15, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The *noise sensitivity* of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is one of its fundamental properties. For noise parameter δ, the noise sensitivity is denoted as $NS_\delta[f]$. This quantity is defined as follows: First, pick $x = (x_1, \ldots, x_n)$ uniformly at random from $\{0,1\}^n$, then pick z by flipping each x_i independently with probability δ. $NS_\delta[f]$ is defined to equal $\Pr[f(x) \neq f(z)]$. Much of the existing literature on noise sensitivity explores the following two directions: (1) Showing that functions with low noise-sensitivity are structured in certain ways. (2) Mathematically showing that certain classes of functions have low noise sensitivity. Combined, these two research directions show that certain classes of functions have low noise sensitivity and therefore have useful structure.

The fundamental importance of noise sensitivity, together with this wealth of structural results, motivates the algorithmic question of approximating $NS_\delta[f]$ given an oracle access to the function f. We show that the standard sampling approach is essentially optimal for general Boolean functions. Therefore, we focus on estimating the noise sensitivity of *monotone* functions, which form an important subclass of Boolean functions, since many functions of interest are either monotone or can be simply transformed into a monotone function (for example the class of *unate* functions consists of all the functions that can be made monotone by reorienting some of their coordinates [22]).

Specifically, we study the algorithmic problem of approximating $NS_\delta[f]$ for monotone f, given the promise that $NS_\delta[f] \geq 1/n^C$ for constant C, and for δ in the range $1/n \leq \delta \leq 1/2$. For such f and δ, we give a randomized algorithm that has query complexity of $O\left(\frac{\min(1,\sqrt{n}\delta \log^{1.5} n)}{NS_\delta[f]} \mathrm{poly}\left(\frac{1}{\epsilon}\right)\right)$ and approximates $NS_\delta[f]$ to within a multiplicative factor of $(1 \pm \epsilon)$. Given the same constraints on f and δ, we also prove a lower bound of $\Omega\left(\frac{\min(1,\sqrt{n}\delta)}{NS_\delta[f] \cdot n^\xi}\right)$ on the query complexity of any algorithm that approximates $NS_\delta[f]$ to within any constant factor, where ξ can be any positive constant. Thus, our algorithm's query complexity is close to optimal in terms of its dependence on n.

We introduce a novel *descending-ascending view* of noise sensitivity, and use it as a central tool for the analysis of our algorithm. To prove lower bounds on query complexity, we develop a technique that reduces computational questions about query complexity to combinatorial questions about the existence of "thin" functions with certain properties.

The existence of such “thin" functions is proved using the probabilistic method. These techniques also yield new lower bounds on the query complexity of approximating other fundamental properties of Boolean functions: the *total influence* and the *bias*.

Thesis Supervisor: Ronitt Rubinfeld
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I am indebted to Ronitt Rubinfeld for suggesting the problem and supervising this work.

I also would like to express my sincere gratitude to my family for their unconditional trust and encouragement.

Contents

List of Figures

Chapter 1

Introduction

Noise sensitivity is a property of any Boolean function $f : \{0,1\}^n \to \{0,1\}$ defined as follows: First, pick $x = (x_1, \ldots, x_n)$ uniformly at random from $\{0,1\}^n$, then pick z by flipping each x_i independently with probability δ. Here δ, the noise parameter, is a given positive constant no greater than $1/2$ (and at least $1/n$ in the interesting cases). With the above distributions on x and z, the noise sensitivity of f, denoted as $NS_\delta[f]$, is defined as follows:

$$NS_\delta[f] \stackrel{\text{def}}{=} \Pr[f(x) \neq f(z)] \tag{1.1}$$

Noise sensitivity was first explicitly defined by Benjamini, Kalai and Schramm in [3], and has been the focus of multiple papers: e.g. [3, 7, 8, 10, 11, 13, 18, 23]. It has been applied to learning theory [4, 7, 8, 9, 11, 12, 16], property testing [1, 2], hardness of approximation [14, 17], hardness amplification [20], theoretical economics and political science[10], combinatorics [3, 13], distributed computing [19] and differential privacy [7]. Multiple properties and applications of noise sensitivity are summarized in [21] and [22]. Much of the existing literature on noise sensitivity explores the following directions: (1) Showing that functions with low noise-sensitivity are structured in certain ways. (2) Mathematically showing that certain classes of functions have low noise sensitivity. Combined, these two research directions show that certain classes of functions have low noise sensitivity and therefore have useful structure.

The fundamental importance of noise sensitivity, together with this wealth of structural

results, motivates the algorithmic question of approximating $NS_\delta[f]$ given an oracle access to the function f. It can be shown that standard sampling techniques require $O\left(\frac{1}{NS_\delta[f]\epsilon^2}\right)$ queries to get a $(1+\epsilon)$-multiplicative approximation for $NS_\delta[f]$. In Appendix B, we show that this is optimal for a wide range of parameters of the problem. Specifically, it cannot be improved by more than a constant when ϵ is a sufficiently small constant, δ satisfies $1/n \leq \delta \leq 1/2$ and $NS_\delta[f]$ satisfies $\Omega\left(\frac{1}{2^n}\right) \leq NS_\delta[f] \leq O(1)$.

It is often the case that data possesses a known underlying structural property which makes the computational problem significantly easier to solve. A natural first such property to investigate is that of monotonicity, as a number of natural function families are made up of functions that are either monotone or can be simply transformed into a monotone function (for example the class of *unate* functions consists of all the functions that can be made monotone by reorienting some of their coordinates [22]). Therefore, we focus on estimating the noise sensitivity of monotone functions.

The approximation of the related quantity of total influence (henceforth just influence) of a monotone Boolean function in this model was previously studied by [25, 24][1]. Influence, denoted by $I[f]$, is defined as n times the probability that a random edge of the Boolean cube (x, y) is *influential*, which means that $f(x) \neq f(y)$. (This latter probability is sometimes referred to as the *average sensitivity*). It was shown in [25, 24] that one can approximate the influence of a monotone function f with only $\tilde{O}\left(\frac{\sqrt{n}}{I[f]\text{poly}(\epsilon)}\right)$ queries, which for constant ϵ beats the standard sampling algorithm by a factor of $\sqrt{n}$, ignoring logarithmic factors.

Despite the fact that the noise sensitivity is closely connected to the influence [21, 22], the noise sensitivity of a function can be quite different from its influence. For instance, for the parity function of all n bits, the influence is n, but the noise sensitivity is $\frac{1}{2}(1-(1-2\delta)^n)$ (such disparities also hold for monotone functions, see for example the discussion of influence and noise sensitivity of the majority function in [22]). Therefore, approximating the influence by itself does not give one a good approximation to the noise sensitivity.

The techniques in [25, 24] also do not immediately generalize to the case of noise

[1] [24] is the journal version of [25] and contains a different algorithm that yields sharper results. However, our algorithmic techniques build on the conference version [25].

sensitivity. The result in [25, 24] is based on the observation that given a descending[2] path on the Boolean cube, at most one edge in it can be influential. Thus, to check if a descending path of any length contains an influential edge, it suffices to check the function values at the endpoints of the path. By sampling random descending paths, [25, 24] show that one can estimate the fraction of influential edges, which is proportional to the influence.

The most natural attempt to relate these path-based techniques with the noise sensitivity is to view it in the context of the following process: first one samples x randomly, then one obtains z by taking a random walk from x by going through all the indices in an arbitrary order and deciding whether to flip each with probability δ. The intermediate values in this process give us a natural path connecting x to z. However, this path is in general not descending, so it can, for example, cross an even number of influential edges, and then the function will have the same value on the two endpoints of this path. This prevents one from immediately applying the techniques from [25, 24].

We overcome this difficulty by introducing our main conceptual contribution: the *descending-ascending view* of noise sensitivity. In the process above, instead of going through all the indices in an arbitrary order, we first go through the indices i for which $x_i = 1$ and only then through the ones for which $x_i = 0$. This forms a path between x and z that has first a descending component and then an ascending component. Although this random walk is more amenable to an analysis using the path-based techniques of [25, 24], there are still non-trivial sampling questions involved in the design and analysis of our algorithm.

An immediate corollary of our result is a query complexity upper bound on estimating the gap between the noise stability of a Boolean function and one. The noise stability of a Boolean function f depends on a parameter ρ and is denoted by $\mathrm{Stab}_\rho[f]$ (for more information about noise stability, see [22]). One way $\mathrm{Stab}_\rho[f]$ can be defined is as the unique quantity satisfying the functional relation $\frac{1}{2}(1 - \mathrm{Stab}_{1-2\delta}[f]) = NS_\delta[f]$ for all δ. This implies that by obtaining an approximation for $NS_\delta[f]$, one also achieves an approximation for $1 - \mathrm{Stab}_{1-2\delta}[f]$.

[2] A path is *descending* if each subsequent vertex in it is dominated by all the previous ones in the natural partial order on the Boolean cube.

1.1 Results

Our main algorithmic result is the following:

Theorem 1 *Let δ be a parameter satisfying:*

$$\frac{1}{n} \leq \delta \leq \frac{1}{\sqrt{n}\log^{1.5} n}$$

Suppose, $f : \{0,1\}^n \to \{0,1\}$ is a monotone function and $NS_\delta[f] \geq \frac{1}{n^C}$ for some constant C.

Then, there is an algorithm that outputs an approximation to $NS_\delta[f]$ to within a multiplicative factor of $(1 \pm \epsilon)$, with success probability at least $2/3$. In expectation, the algorithm makes $O\left(\frac{\sqrt{n}\delta\log^{1.5} n}{NS_\delta[f]\epsilon^3}\right)$ queries to the function. Additionally, it runs in time polynomial in n.

Note that computing noise-sensitivity using standard sampling[3] requires $O\left(\frac{1}{NS_\delta[f]\epsilon^2}\right)$ samples. Therefore, for a constant ϵ, we have the most dramatic improvement if $\delta = \frac{1}{n}$, in which case, ignoring constant and logarithmic factors, our algorithm outperforms standard sampling by a factor of $\sqrt{n}$.

As in [25], our algorithm requires that the noise sensitivity of the input function f is larger than a specific threshold $1/n^C$. Our algorithm is not sensitive to the value of C as long as it is a constant, and we think of $1/n^C$ as a rough initial lower bound known in advance.

We next give lower bounds for approximating three different parameters of monotone Boolean functions: the bias, the influence and the noise sensitivity. A priori, it is not clear what kind of lower bounds one could hope for. Indeed, determining whether a given function is the all-zeros function requires $\Omega(2^n)$ queries in the general function setting, but only 1 query (of the all-ones input), if the function is promised to be monotone. Nevertheless, we show that such a dramatic improvement for approximating these quantities is not possible.

[3]Standard sampling refers to the algorithm that picks $O\left(\frac{1}{NS_\delta[f]\epsilon^2}\right)$ pairs x and z as in the definition of noise sensitivity and computes the fraction of pairs for which $f(x) \neq f(z)$.

For monotone functions, we are not aware of previous lower bounds on approximating the bias or noise sensitivity. Our lower bound on approximating influence is not comparable to the lower bounds in [25, 24], as we will elaborate shortly.

We now state our lower bound for approximating the noise sensitivity. Here and everywhere else, to "reliably distinguish" means to distinguish with probability at least $2/3$.

Theorem 2 *For all constants C_1 and C_2 satisfying $C_1 - 1 > C_2 \geq 0$, for an infinite number of values of n the following is true: For all δ satisfying $1/n \leq \delta \leq 1/2$, given a monotone function $f : \{0,1\}^n \to \{0,1\}$, one needs at least $\Omega\left(\frac{n^{C_2}}{e^{\sqrt{C_1 \log n/2}}}\right)$ queries to reliably distinguish between the following two cases: (i) f has noise sensitivity between $\Omega(1/n^{C_1+1})$ and $O(1/n^{C_1})$ and (ii) f has noise sensitivity larger than $\Omega(\min(1, \delta\sqrt{n})/n^{C_2})$.*

Remark 1 *For any positive constant ξ, we have that $e^{\sqrt{C_1 \log n/2}} \leq n^{\xi}$.*

Remark 2 *The range of the parameter δ can be divided into two regions of interest. In the region $1/n \leq \delta \leq 1/(\sqrt{n}\log n)$, the algorithm from Theorem 1 can distinguish the two cases above with only $\tilde{O}(n^{C_2})$ queries. Therefore its query complexity is optimal up to a factor of $\tilde{O}(e^{\sqrt{C_1 \log n/2}})$. Similarly, in the region $1/(\sqrt{n}\log n) \leq \delta \leq 1/2$, the standard sampling algorithm can distinguish the two distributions above with only $\tilde{O}(n^{C_2})$ queries. Therefore in this region of interest, standard sampling is optimal up to a factor of $\tilde{O}(e^{\sqrt{C_1 \log n/2}})$.*

We define the *bias* of a Boolean function as $B[f] \stackrel{\text{def}}{=} \Pr[f(x) = 1]$, where x is chosen uniformly at random from $\{0,1\}^n$. It is arguably the most basic property of a Boolean function, so we consider the question of how quickly it can be approximated for monotone functions. To approximate the bias up to a multiplicative factor of $1 \pm \epsilon$ using standard sampling, one needs $O(1/(B[f]\epsilon^2))$ queries. We obtain a lower bound for this task similar to the previous theorem:

Theorem 3 *For all constants C_1 and C_2 satisfying $C_1 - 1 > C_2 \geq 0$, for an infinite number of values of n the following is true: Given a monotone function f: $\{0,1\}^n \to \{0,1\}$,*

one needs at least $\Omega\left(\frac{n^{C_2}}{e^{\sqrt{C_1 \log n/2}}}\right)$ *queries to reliably distinguish between the following two cases: (i)* f *has bias of* $\Theta(1/n^{C_1})$ *(ii)* f *has bias larger than* $\Omega(1/n^{C_2})$.

Finally we prove a lower bound for approximating influence:

Theorem 4 *For all constants* C_1 *and* C_2 *satisfying* $C_1 - 1 > C_2 \geq 0$*, for an infinite number of values of* n *the following is true: Given a monotone function* $f\colon \{0,1\}^n \to \{0,1\}$*, one needs at least* $\Omega\left(\frac{n^{C_2}}{e^{\sqrt{C_1 \log n/2}}}\right)$ *queries to reliably distinguish between the following two cases: (i)* f *has influence between* $\Omega(1/n^{C_1})$ *and* $O(n/n^{C_1})$ *(ii)* f *has influence larger than* $\Omega(\sqrt{n}/n^{C_2})$.

This gives us a new sense in which the algorithm family in [25, 24] is close to optimal, because for a function f with influence $\Omega(\sqrt{n}/n^{C_2})$ this algorithm makes $\tilde{O}(n^{C_2})$ queries to estimate the influence up to any constant factor.

Our lower bound is incomparable to the lower bound in [25], which makes the stronger requirement that $I[f] \geq \Omega(1)$, but gives a bound that is only a polylogarithmic factor smaller than the runtime of the algorithm in [25, 24]. There are many possibilities for algorithmic bounds that were compatible with the lower bound in [25, 24], but are eliminated with our lower bound. For instance, prior to this work, it was conceivable that an algorithm making as little as $O(\sqrt{n})$ queries could give a constant factor approximation to the influence of **any** monotone input function whatsoever. Our lower bound shows that not only is this impossible, no algorithm that makes $O(n^{C_2})$ queries for any constant C_2 can accomplish this either.

1.2 Algorithm overview

Here, we give the algorithm in Theorem 1 together with the subroutines it uses. Additionally, we give an informal overview of the proof of correctness and the analysis of running time and query complexity, which are presented in Section 3.

First of all, recall that $NS_\delta[f] = \Pr[f(x) \neq f(z)]$ by Equation 1.1. Using a standard pairing argument, we argue that $NS_\delta[f] = 2 \cdot \Pr[f(x) = 1 \wedge f(z) = 0]$. In other words, we can focus only on the case when the value of the function flips from one to zero.

Process D

1. Pick x uniformly at random from $\{0,1\}^n$. Let S_0 be the set of indexes i for which $x_i = 0$, and conversely let S_1 be the rest of indexes.
2. **Phase 1:** go through all the indexes in S_1 in a random order, and flip each with probability δ. Form the descending path P_1 from all the intermediate results. Call the endpoint y.
3. **Phase 2:** start at y, and flip each index in S_0 with probability δ. As before, all the intermediate results form an ascending path P_2, which ends in z.
4. Output P_1, P_2, x, y and z.

Figure 1-1: The noise process

We introduce the *descending-ascending view of noise sensitivity* (described more formally in Section 3.1), which, roughly speaking, views the noise process as decomposed into a first phase that operates only on the locations in x that are 1, and a second phase that operates only on the locations in x that are set to 0. Formally, we describe the noise process in Figure 1.2. This process gives us a path from x to z that can be decomposed into two segments, such that the first part, P_1, descends in the hypercube, and the second part P_2 ascends in the hypercube.

Since f is monotone, for $f(x) = 1$ and $f(z) = 0$ to be the case, it is necessary, though not sufficient, that $f(x) = 1$ and $f(y) = 0$, which happens whenever P_1 hits an influential edge. Therefore we break the task of estimating the probability of $f(x) \neq f(z)$ into computing the product of:

- The probability that P_1 hits an influential edge, specifically, the probability that $f(x) = 1$ and $f(y) = 0$, which we refer to as p_A.
- The probability that P_2 does not hit any influential edge, given that P_1 hits an influential edge: specifically, the probability that given $f(x) = 1$ and $f(y) = 0$, it is the case that $f(z) = 0$. We refer to this probability as p_B.

The above informal definitions of p_A and p_B ignore some technical complications. Specifically, the impact of certain “bad events" is considered in our analysis. We redefine p_A and

p_B precisely in Section 3.1.3.

To define those bad events, we use the following two values, which we reference in our algorithms: t_1 and t_2. Informally, t_1 and t_2 have the following intuitive meaning. A typical vertex x of the hypercube has Hamming weight $L(x)$ between $n/2 - t_1$ and $n/2 + t_1$. A typical Phase 1 path from process D will have length at most t_2. To achieve this, we assign $t_1 \stackrel{\text{def}}{=} \eta_1 \sqrt{n \log n}$ and $t_2 \stackrel{\text{def}}{=} n\delta(1 + 3\eta_2 \log n)$, where η_1 and η_2 are certain constants.

We also define M to be the set of edges $e = (v_1, v_2)$, for which both $L(v_1)$ and $L(v_2)$ are between and $n/2 - t_1$ and $n/2 + t_1$. Most of the edges in the hypercube are in M, which is used by our algorithm and the run-time analysis.

Our analysis requires that only $\delta \leq 1/(\sqrt{n} \log^{1.5} n)$ as in the statement of Theorem 1, however the utility of the ascending-descending view can be most clearly motivated when $\delta \leq 1/(\sqrt{n} \log^2 n)$. Specifically, given that $\delta \leq 1/(\sqrt{n} \log^2 n)$, it is the case that t_2 will be shorter than $O(\sqrt{n}/\log n)$. Therefore, typically, the path P_1 is also shorter than $O(\sqrt{n}/\log n)$. Similar short descending paths on the hypercube have been studied before: In [25], paths of such lengths were used to estimate the number of influential edges by analyzing the probability that a path would hit such an edge. One useful insight given by [25] is that the probability of hitting almost every single influential edge is roughly the same.

However, the results in [25] cannot be immediately applied to analyze P_1, because (i) P_1 does not have a fixed length, but rather its lengths form a probability distribution, (ii) this probability distribution also depends on the starting point x of P_1. We build upon the techniques in [25] to overcome these difficulties, and prove that again, roughly speaking, for almost every single influential edge, the probability that P_1 hits it depends very little on the location of the edge, and our proof also computes this probability. This allows us to prove that $p_A \approx \delta I[f]/2$. Then, using the algorithm in [25] to estimate $I[f]$, we estimate p_A.

Regarding p_B, we estimate it by approximately sampling paths P_1 and P_2 that would arise from process D, conditioned on that P_1 hits an influential edge. To that end, we first sample an influential edge e that P_1 hits. Since P_1 hits almost every single influential edge

Algorithm $\mathcal{A}$ (given oracle access to a monotone function $f : \{0,1\}^n \to \{0,1\}$ and a parameter ϵ)

1. Assign $w = \frac{\epsilon}{3100\eta_1}\sqrt{\frac{n}{\log n}}$
2. Pick x uniformly at random from $\{0,1\}^n$.
3. Perform a descending walk P_1 downwards in the hypercube starting at x. Stop at a vertex y either after w steps, or if you hit the all-zeros vertex. Query the value of f only at the endpoints x and y of this path.
4. If $f(x) = f(y)$ output FAIL.
5. If $f(x) \neq f(y)$ perform a binary search on the path P_1 and find an influential edge e_{inf}.
6. If $e_{inf} \in M$ return e_{inf}. Otherwise output FAIL.

Figure 1-2: Algorithm for sampling an influential edge

with roughly the same probability, we do it by sampling e approximately uniformly from among influential edges. For the latter task, we build upon the result in [25] as follows: As we have already mentioned, the algorithm in [25] samples descending paths of a fixed length to estimate the influence. For those paths that start at an x for which $f(x) = 1$ and end at a z for which $f(z) = 0$, we add a binary search step in order to locate the influential edge e that was hit by the path.

Thus, we have the algorithm $\mathcal{A}$ in Figure 1.2, which takes oracle access to a function f and an approximation parameter ϵ as input. In the case of success, it outputs an influential edge that is roughly uniformly distributed:

Finally, once we have obtained a roughly uniformly random influential edge e, we sample a path P_1 from among those that hit it. Interestingly, we show that this can be accomplished by a simple exponential time algorithm that makes no queries to f. However, the constraint on the run-time of our algorithm forces us to follow a different approach:

An obvious way to try to quickly sample such a path is to perform two random walks of lengths w_1 and w_2 in opposite directions from the endpoints of the edge, and then concatenate them into one path. However, to do this, one needs to somehow sample the lengths w_1

Algorithm $\mathcal{W}$ (given an edge $e \stackrel{\text{def}}{=} (v_1, v_2)$ so $v_2 \preceq v_1$)

1. Pick an integer l uniformly at random among the integers in $[L(v_1), L(v_1) + t_2 - 1]$. Pick a vertex x randomly at level l.
2. As in phase 1 of the noise sensitivity process, traverse in random order through the indices of x and for each index that equals to one, flip it with probability δ. The intermediate results form a path P_1, and we call its endpoint y.
3. If P_1 does not intersect Λ_e go to step 1.
4. Otherwise, output $w_1 = L(x) - L(v_1)$ and $w_2 = L(v_2) - L(y)$.

Figure 1-3: Algorithm $\mathcal{W}$

and w_2. This problem is not trivial, since longer descending paths are more likely to hit an influential edge, which biases the distribution of the path lengths towards longer ones.

To generate w_1 and w_2 according to the proper distribution, we first sample a path P_1 hitting any edge at the same *layer*[4] Λ_e as e. We accomplish this by designing an algorithm that uses rejection sampling. The algorithm samples short descending paths from some conveniently chosen distribution, until it gets a path hitting the desired layer.

We now describe the algorithm in more detail. Recall that we use $L(x)$ to denote the Hamming weight of x, which equals the number of indices i on which $x_i = 1$, and we use the symbol Λ_e to denote the whole layer of edges that have the same endpoint levels as e. The algorithm $\mathcal{W}$ described in Figure 1.2 takes an influential edge e as an input and samples the lengths w_1 and w_2. Recall that t_2 has a technical role and is defined to be equal $n\delta(1 + 3\eta_2 \log n)$, where η_2 is a certain constant. t_2 is chosen to be long enough that it is longer than most paths P_1, but short enough to make the sampling in $\mathcal{W}$ efficient. Since the algorithm involves short descending paths, we analyze this algorithm building upon the techniques we used to approximate p_A.

After obtaining a random path going through the same layer as e, we show how to transform it using the symmetries of the hypercube, into a a random path P_1 going through

[4]We say that edges e_1 and e_2 are on the same layer if and only if their endpoints have the same Hamming weights. We denote the layer an edge e belongs to as Λ_e.

Algorithm $\mathcal{B}$ (given an influential edge $e \stackrel{\text{def}}{=} (v_1, v_2)$ so $v_2 \preceq v_1$)

1. Use $\mathcal{W}(e)$ to sample w_1 and w_2.
2. Perform an ascending random walk of length w_1 starting at v_1 and call its endpoint x. Similarly, perform a descending random walk starting at v_2 of length w_2, call its endpoint y.
3. Define P_1 as the descending path that results between x and y by concatenating the two paths from above, oriented appropriately, and edge e.
4. Define P_2 just as in phase 2 of our process starting at y. Consider in random order all the zero indices y has in common with x and flip each with probability δ.
5. Return P_1 ,P_2, x, y and z.

Figure 1-4: Algorithm $\mathcal{B}$

e itself. Additionally, given the endpoint of P_1, we sample the path P_2 just as in the process D.

Formally, we use the algorithm $\mathcal{B}$ shown in Figure 1.2 that takes an influential edge e and returns a descending path P_1 that goes through e and an adjacent ascending path P_2, together with the endpoints of these paths. We then use sampling to estimate which fraction of the paths P_2 continuing these P_1 paths does not hit an influential edge. This allows us to estimate p_B, which, combined with our estimate for p_A, gives us an approximation for $NS_\delta[f]$.

Formally, we put all the previously defined subroutines together into the randomized algorithm in Figure 1.2 that takes oracle access to a function f together with an approximation parameter ϵ and outputs an approximation to $NS_\delta[f]$.

1.3 Lower bound techniques

We use the same technique to lower bound the query complexity of approximating any of the following three quantities: the noise sensitivity, influence and bias.

For concreteness, let us first focus on approximating the bias. Recall that one can

Algorithm for estimating noise sensitivity. (given oracle access to a monotone function $f\colon \{0,1\}^n \to \{0,1\}$, and a parameter ϵ)

1. Using the algorithm from [25] as described in Theorem 5, compute an approximation to the influence of f to within a multiplicative factor of $(1 \pm \epsilon/33)$. This gives us $\tilde{I}$.
2. Compute $\tilde{p}_A := \delta\tilde{I}/2$.
3. Initialize $\alpha := 0$ and $\beta := 0$. Repeat the following until $\alpha = \frac{768 \ln 200}{\epsilon^2}$.
   - Use algorithm $\mathcal{A}$ from Lemma 3.3.1 repeatedly to successfully sample an edge e.
   - From Lemma 3.4.3 use the algorithm $\mathcal{B}$, giving it e as input, and sample P_1, P_2, x, y and z.
   - If it is the case that $f(x) = 1$ and $f(z) = 0$, then $\alpha := \alpha + 1$.
   - $\beta := \beta + 1$.
4. Set $\tilde{p}_B = \frac{\alpha}{\beta}$.
5. Return $2\tilde{p}_A\tilde{p}_B$.

Figure 1-5: Algorithm for estimating noise sensitivity.

distinguish the case where the bias is 0 from the bias being $1/2^n$ using a single query. Nevertheless, we show that for the most part, no algorithm for estimating the bias can do much better than the random sampling approach.

We construct two probability distributions D_1^B and D_2^B that are relatively hard to distinguish but have drastically different biases. To create them, we fix some threshold l_0 and then construct a special monotone function F^B, which has the following two properties: (1) It has a high bias. (2) It equals to one on only a relatively small fraction of points on the level l_0. We refer to functions satisfying (2) as "thin" functions. We will explain later how to obtain such a function F^B. We pick a function from D_2^B by taking F^B, randomly permuting the indices of its input, and finally "truncating" it by setting it to one on all values x for on levels higher than l_0.

We form D_1^B even more simply. We take the all-zeros function and truncate it at the same threshold l_0. The threshold l_0 is chosen in a way that this function in D_1^B has a sufficiently small bias. Thus D_1^B consists of only a single function.

The purpose of truncation is to prevent a distinguisher from gaining information by accessing the values of the function on the high-level vertices of the hypercube. Indeed, if there was no truncation, one could tell whether they have access to the all-zeros function by simply querying it on the all-ones input. Since F^B is monotone, if it equals to one on at least one input, then it has to equal one on the all-ones input.

The proof has two main lemmas: The first one is computational and says that if F^B is "thin" then D_1^B and D_2^B are hard to reliably distinguish. To prove the first lemma, we show that one could transform any adaptive algorithm for distinguishing D_1^B from D_2^B into an algorithm that is just as effective, is non-adaptive and queries points only on the layer l_0.

To show this, we observe that, because of truncation, distinguishing a function in D_2^B from a function in D_1^B is in a certain sense equivalent to finding a point with level at most l_0 on which the given function evaluates to one. We argue that for this setting, adaptivity does not help. Additionally, if $x \preceq y$ and both of them have levels at most l_0 then, since f is monotone, $f(x) = 1$ implies that $f(y) = 1$ (but not necessarily the other way around). Therefore, for finding a point on which the function evaluates to one, it is never more useful

to query x instead of y.

Once we prove that no algorithm can do better than a non-adaptive algorithm that only queries points on the level l_0, we use a simple union bound to show that any such algorithm cannot be very effective for distinguishing our distributions.

Finally, to construct F^B, we need to show that there exist functions that are "thin" and simultaneously have a high bias. This is a purely combinatorial question and is proven in our second main lemma. We build upon Talagrand random functions that were first introduced in [26]. In [18] it was shown that they are very sensitive to noise, which was applied for property testing lower bounds [2]. A Talagrand random DNF consists of $2^{\sqrt{n}}$ clauses of $\sqrt{n}$ indices chosen randomly with replacement. We modify this construction by picking the indices without replacement and generalize it by picking $2^{\sqrt{n}}/n^{C_2}$ clauses, where C_2 is a non-negative constant. We show that these functions are "thin", so they are appropriate for our lower bound technique.

"Thinness" allows us to conclude that D_1^B and D_2^B are hard to distinguish from each other. We then prove that they have drastically different biases. We do the latter by employing the probabilistic method and showing that in expectation our random function has a large enough bias. We handle influence and noise sensitivity analogously, specifically by showing that that as we pick fewer clauses, the expected influence and noise sensitivity decrease proportionally. We prove this by dividing the points, where one of these random functions equals to one, into two regions: (i) the region where only one clause is true and (ii) a region where more than one clause is true. Roughly speaking, we show that the contribution from the points in (i) is sufficient to obtain a good lower bound on the influence and noise sensitivity.

1.4 Possibilities of improvement?

In [24] (which is the journal version of [25]), it was shown that using the chain decomposition of the hypercube, one can improve the run-time of the algorithm to $O\left(\frac{\sqrt{n}}{\epsilon^2 I[f]}\right)$ and also improve the required lower bound on $I[f]$ to be $I[f] \geq \exp(-c_1\epsilon^2 n + c_2 \log(n/\epsilon))$ for

some constant c_1 and c_2 (it was $I[f] \geq 1/n^C$ for any constant C in [25]). Additionally, the algorithm itself was considerably simplified.

A hope is that techniques based on the chain decomposition could help improve the algorithm in Theorem 1. However, it is not clear how to generalize our approach to use these techniques, since the ascending-descending view is a natural way to express noise sensitivity in terms of random walks, and it is not obvious whether one can replace these walks with chains of the hypercube.

Chapter 2

Preliminaries

2.1 Definitions

2.1.1 Fundamental definitions and lemmas pertaining to the hypercube.

Definition 1 *We refer to the poset over $\{0,1\}^n$ as the n-**dimensional hypercube**, viewing the domain as vertices of a graph, in which two vertices are connected by an edge if and only if the corresponding elements of $\{0,1\}^n$ differ in precisely one index. For $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ in $\{0,1\}^n$, we say that $x \preceq y$ if and only if for all i in $[n]$ it is the case that $x_i \leq y_i$.*

Definition 2 *The **level of a vertex** x on the hypercube is the hamming weight of x, or in other words number of 1-s in x. We denote it by $L(x)$.*

We define the set of edges that are in the same “layer" of the hypercube as a given edge:

Definition 3 *For an arbitrary edge e suppose $e = (v_1, v_2)$ and $v_2 \preceq v_1$. We denote Λ_e to be the set of all edges $e' = (v_1', v_2')$, so that $L(v_1) = L(v_1')$ and $L(v_2) = L(v_2')$.*

The size of Λ_e is $L(v_1)\binom{n}{L(v_1)}$. The concept of Λ_e will be useful because we will deal with paths that are symmetric with respect to change of coordinates, and these have an equal probability of hitting any edge in Λ_e.

As we view the hypercube as a graph, we will often refer to paths on it. By referring to a path P we will, depending on the context, refer to its set of vertices or edges.

Definition 4 *We call a path **descending** if for every pair of consecutive vertices v_i and v_{i+1}, it is the case that $v_{i+1} \prec v_i$. Conversely, if the opposite holds and $v_i \prec v_{i+1}$, we call a path **ascending**. We consider an empty path to be vacuously both ascending and descending. We define the length of a path to be the number of edges in it, and denote it by $|P|$. We say we **take a descending random walk of length** w **starting at** x, if we pick a uniformly random descending path of length w starting at x.*

Descending random walks over the hyper-cube were used in an essential way in [25] and were central for the recent advances in monotonicity testing algorithms [5, 6, 15].

Lemma 2.1.1 (Hypercube Continuity Lemma) *Suppose n is a sufficiently large positive integer, C_1 is a constant and we are given l_1 and l_2 satisfying:*

$$\frac{n}{2} - \sqrt{C_1 n \log(n)} \leq l_1 \leq l_2 \leq \frac{n}{2} + \sqrt{C_1 n \log(n)}$$

If we denote $C_2 \stackrel{\text{def}}{=} \frac{1}{10\sqrt{C_1}}$, then for any ξ satisfying $0 \leq \xi \leq 1$, if it is the case that $l_2 - l_1 \leq C_2 \xi \sqrt{\frac{n}{\log(n)}}$, then, for large enough n, it is the case that $1 - \xi \leq \frac{\binom{n}{l_1}}{\binom{n}{l_2}} \leq 1 + \xi$

Proof: See Appendix C, Section C.2. ■

2.1.2 Fundamental definitions pertaining to Boolean functions

We define monotone functions over the n-dimensional hypercube:

Definition 5 *Let f be a function $\{0,1\}^n \to \{0,1\}^n$. We say that f is **monotone** if for any x and y in $\{0,1\}^n$, $x \preceq y$ implies that $f(x) \leq f(y)$*

Influential edges, and influence of a function are defined as follows:

Definition 6 *An edge (x, y) in the hypercube is called **influential** if $f(x) \neq f(y)$. Additionally, we denote the set of all influential edges in the hypercube as E_I.*

Definition 7 *The **influence** of function $f\colon \{0,1\}^n \to \{0,1\}$, denoted by $I[f]$ is:*

$$I[f] \stackrel{\text{def}}{=} n \cdot \Pr_{x \in_R \{0,1\}^n, i \in_R [n]}[f(x) \neq f(x^{\oplus i})]$$

Where $x^{\oplus i}$ is x with its i-th bit flipped.[1] *Equivalently, the influence is n times the probability that a random edge is influential. Since there are $n \cdot 2^{n-1}$ edges, then $|E_I| = 2^{n-1} I[f]$.*

Definition 8 *Let δ be a parameter and let x be selected uniformly at random from $\{0,1\}^n$. Let $z \in \{0,1\}^n$ be defined as follows:*

$$z_i = \begin{cases} x_i & \text{with probability } 1-\delta \\ 1 - x_i & \text{with probability } \delta \end{cases}$$

*We denote this distribution of x and z by T_δ. Then we define the **noise sensitivity** of f as:*

$$NS_\delta[f] \stackrel{\text{def}}{=} \Pr_{(x,z) \in_R T_\delta}[f(x) \neq f(z)]$$

Observation 2.1.2 *For every pair of vertices a and b, the probability that for a pair x, z drawn from T_δ, it is the case that $(x,z) = (a,b)$, is equal to the probability that $(x,z) = (b,a)$. Therefore,*

$$\Pr[f(x) = 0 \wedge f(z) = 1] = \Pr[f(x) = 1 \wedge f(z) = 0]$$

Hence:

$$NS_\delta[f] = 2 \cdot \Pr[f(x) = 1 \wedge f(z) = 0]$$

2.1.3 Influence estimation.

To estimate the influence, standard sampling would require $O\left(\frac{n}{I[f]\epsilon^2}\right)$ samples. However, from [25] we have:

[1] We use the symbol $\in_R$ to denote, depending on the type of object the symbol is followed by: (i) Picking a random element from a probability distribution. (ii) Picking a uniformly random element from a set (iii) Running a randomized algorithm and taking the result.

Theorem 5 *There is an algorithm that approximates $I[f]$ to within a multiplicative factor of $(1 \pm \epsilon)$ for a monotone $f\colon \{0,1\}^n \to \{0,1\}$. The algorithm requires that $I[f] \geq 1/n^{C'}$ for a constant C' that is given to the algorithm. It outputs a good approximation with probability at least 0.99 and in expectation requires $O\left(\frac{\sqrt{n}\log(n/\epsilon)}{I[f]\epsilon^3}\right)$ queries. Additionally, it runs in time polynomial in n.*

2.1.4 Bounds for ϵ and $I[f]$

The following observation allows us to assume that without loss of generality ϵ is not too small. A similar technique was also used in [25].

Observation 2.1.3 *When $\epsilon < O(\sqrt{n}\delta\log^{1.5}(n))$ there is a simple algorithm that accomplishes the desired query complexity of $O\left(\frac{\sqrt{n}\delta\log^{1.5}(n)}{NS_\delta[f]\epsilon^3}\right)$. Namely, this can be done by the standard sampling algorithm that requires only $O\left(\frac{1}{NS_\delta[f]\epsilon^2}\right)$ samples. Thus, since we can handle the case when $\epsilon < O(\sqrt{n}\delta\log^{1.5}(n))$, we focus on the case when $\epsilon \geq H\sqrt{n}\delta\log^{1.5}(n) \geq H\frac{\log^{1.5} n}{\sqrt{n}}$, for any constant H.*

Additionally, throughout the paper whenever we need it, we will without loss of generality assume that ϵ is smaller than a sufficiently small positive constant.

We will also need a known lower bound on influence:

Observation 2.1.4 *For any function $f\colon \{0,1\}^n \to \{0,1\}$ and $\delta \leq 1/2$ it is the case that:*

$$NS_\delta[f] \leq \delta I[f]$$

Therefore it is the case that $I[f] \geq \frac{1}{n^C}$.

A very similar statement is proved in [18] and for completeness we prove it in Appendix A.

Chapter 3

An improved algorithm for small δ

In this section we give an improved algorithm for small δ, namely $1/n \leq \delta \leq 1/(\sqrt{n}\log n)$. We begin by describing the descending-ascending view on which the algorithm is based.

3.1 Descending-ascending framework

3.1.1 The descending-ascending process.

As we mentioned, it is be useful to view noise sensitivity in the context of the noise process in Figure 3.1.1. We will use the following notation for probabilities of various events: for an algorithm or a random process X we will use the expression $\Pr_X[]$ to refer to the random variables we defined in the context of this process or algorithm. It will often be the case that the same symbol, say x, will refer to different random random variables in the context of different random processes, so $\Pr_X[x = 1]$ might not be the same as $\Pr_Y[x = 1]$.

By inspection, x and z are distributed identically in D as in T_δ. Therefore from Observation 2.1.2:

$$NS_\delta[f] = 2 \cdot \Pr_D[f(x) = 1 \wedge f(z) = 0]$$

Observation 3.1.1 *Since the function is monotone, if $f(x) = 1$ and $f(z) = 0$, then it has to be that $f(y) = 0$.*

Process D

1. Pick x uniformly at random from $\{0,1\}^n$. Let S_0 be the set of indexes i for which $x_i = 0$, and conversely let S_1 be the rest of indexes.
2. **Phase 1:** go through all the indexes in S_1 in a random order, and flip each with probability δ. Form the descending path P_1 from all the intermediate results. Call the endpoint y.
3. **Phase 2:** start at y, and flip each index in S_0 with probability δ. As before, all the intermediate results form an ascending path P_2, which ends in z.
4. Output P_1, P_2, x, y and z.

Figure 3-1: The noise process (restated)

Now we define the probability that a Phase 1 path starting somewhere at level l makes at least w steps downwards:

Definition 9 *For any l and w in $[n]$ we define $Q_{l,w}$ as follows:*

$$Q_{l;w} \stackrel{\text{def}}{=} \Pr_D[\left|P_1\right| \geq w | L(x) = l]$$

This notation is useful when one wants to talk about the probability that a path starting on a particular vertex hits a specific level.

3.1.2 Defining bad events

In this section, we give the parameters that we use to determine the lengths of our walks, as well as the "middle" of the hypercube.

Define the following values:

$$t_1 \stackrel{\text{def}}{=} \eta_1 \sqrt{n \log n} \qquad\qquad t_2 \stackrel{\text{def}}{=} n\delta(1 + 3\eta_2 \log n)$$

Here η_1 and η_2 are large enough constants. Taking $\eta_1 = \sqrt{C}+4$ and $\eta_2 = C+2$ is sufficient for our purposes (recall that we were promised that $NS_\delta[f] \geq 1/n^C$ for a constant C).

Informally, t_1 and t_2 have the following intuitive meaning. A typical vertex x of the hypercube has $L(x)$ between $n/2 - t_1$ and $n/2 + t_1$. A typical Phase 1 path from process D will have length at most t_2.

We define the "middle edges" M as the following set of edges:

$$M \stackrel{\text{def}}{=} \{e = (v_1, v_2) : \frac{n}{2} - t_1 \leq L(v_2) \leq L(v_1) \leq \frac{n}{2} + t_1\}$$

Denote by $\overline{M}$ the rest of the edges.

We define two bad events in context of D, such that when neither of these events happen, we can show that the output has certain properties. The first one happens roughly when P_1 (from x to y, as defined by Process D) is much longer than it should be in expectation, and the second one happens when P_1 crosses one of the edges that are too far from the middle of the hypercube, which could happen because P_1 is long or because of a starting point that is far from the middle. More specifically:

- E_1 happens when both of the following hold (i) P_1 crosses an edge $e \in E_I$ and (ii) denoting $e = (v_1, v_2)$, so that $v_2 \preceq v_1$, it is the case that $L(x) - L(v_1) \geq t_2$.
- E_2 happens when P_1 contains an edge in $E_I \cap \overline{M}$.

While defining E_1 we want two things from it. First of all, we want its probability to be upper-bounded easily. Secondly, we want it not to complicate the sampling of paths in Lemma 3.4.1. There exists a tension between these two requirements, and as a result the definition of E_1 is somewhat convoluted.

3.1.3 Defining p_A, p_B

We define:

$$p_A \stackrel{\text{def}}{=} \Pr_D[f(x) = 1 \wedge f(y) = 0 \wedge \overline{E_1} \wedge \overline{E_2}]$$
$$p_B \stackrel{\text{def}}{=} \Pr_D[f(z) = 0 | f(x) = 1 \wedge f(y) = 0 \wedge \overline{E_1} \wedge \overline{E_2}]$$

Ignoring the bad events, P_A is the probability that P_1 hits an influential edge, and P_B is the probability that given that P_1 hits an influential edge P_2 does not hit an influential edge. From Observation (3.1.1), if and only if these two things happen, it is the case that $f(x) = 1$ and $f(z) = 0$. From this fact and the laws of conditional probabilities we have:

$$\Pr_D[f(x) = 1 \wedge f(z) = 0 \wedge \overline{E_1} \wedge \overline{E_2}] = \Pr_D[f(x) = 1 \wedge f(y) = 0 \wedge f(z) = 0 \wedge \overline{E_1} \wedge \overline{E_2}] = p_A p_B \tag{3.1}$$

We can consider for every individual edge e in $M \cap E_I$ the probabilities:

$$p_e \stackrel{\text{def}}{=} \Pr_D[e \in P_1 \wedge \overline{E_1} \wedge \overline{E_2}]$$

$$q_e \stackrel{\text{def}}{=} \Pr_D[f(x) = 1 \wedge f(z) = 0 | e \in P_1 \wedge \overline{E_1} \wedge \overline{E_2}] = \Pr_D[f(z) = 0 | e \in P_1 \wedge \overline{E_1} \wedge \overline{E_2}]$$

The last equality is true because $e \in P_1$ already implies $f(x) = 1$. Informally and ignoring the bad events again, p_e is the probability that $f(x) = 1$ and $f(y) = 0$ **because** P_1 hits e and not some other influential edge. Similarly, q_e is the probability $f(x) = 1$ and $f(z) = 0$ given that P_1 hits specifically e.

Since f is monotone, P_1 can hit at most one influential edge. Therefore, the events of P_1 hitting different influential edges are disjoint. Using this, Equation (3.1) and the laws of conditional probabilities we can write:

$$p_A = \sum_{e \in E_I \cap M} p_e \tag{3.2}$$

Furthermore, the events that P_1 hits a given influential edge and then P_2 does not hit any are also disjoint for different influential edges. Therefore, analogous to the previous equation we can write:

$$p_A p_B = \Pr_D[(f(x) = 1) \wedge (f(z) = 0) \wedge \overline{E_1} \wedge \overline{E_2}] = \sum_{e \in E_I \cap M} p_e q_e \tag{3.3}$$

3.1.4 Bad events can be "ignored"

In the following proof, we will need to consider probability distributions in which bad events do not happen. For the most part, conditioning on the fact that bad events do not happen changes little in the calculations. In this subsection, we prove lemmas that allow us to formalize these claims.

The following lemma suggests that almost all influential edges are in M.

Observation 3.1.2 *It is the case that:*

$$\left(1 - \frac{\epsilon}{310}\right)|E_I| \leq |M \cap E_I| \leq |E_I|$$

Proof: This is the case, because:

$$|\overline{M} \cap E_I| \leq |\overline{M}| \leq 2^n n \cdot 2\exp(-2\eta_1^2 \log(n))$$
$$= 2^{n-1} \cdot 4/n^{2\eta_1^2 - 1} \leq 2^{n-1} I[f]/n = |E_I|/n \leq \frac{\epsilon}{310}|E_I|$$

The second inequality is the Hoeffding bound, then we used Observations 2.1.4 and 2.1.3. ■

Lemma 3.1.3 *We proceed to prove that ignoring these bad events does not distort our estimate for $NS_\delta[f]$. It is the case that:*

$$p_A p_B \leq \frac{1}{2} NS_\delta[f] \leq \left(1 + \frac{\epsilon}{5}\right) p_A p_B$$

Proof: See Appendix C, Section C.3. ■

3.2 Main lemmas

Here we prove the correctness and run-time of the main subroutines used in our algorithm for estimating noise sensitivity. For completeness, we will repeat all the algorithms.

3.3 Lemmas about descending paths hitting influential edges

Here we prove two lemmas that allow the estimation of the probability that a certain descending random walk hits an influential edge. As we mentioned in the introduction, except for the binary search step, the algorithm in Lemma 3.3.1 is similar to the algorithm in [25]. In principle, we could have carried out much of the analysis of the algorithm in Lemma 3.3.1 by referencing an equation in [25]. However, for subsequent lemmas, including Lemma 3.3.2, we build on the application of the Hypercube Continuity Lemma to the analysis of random walks on the hypercube. Thus, we give a full analysis of the algorithm in Lemma 3.3.1 here, in order to demonstrate how the Hypercube Continuity Lemma (Lemma 2.1.1) can be used to analyze random walks on the hypercube, before handling the more complicated subsequent lemmas, including Lemma 3.3.2.

Lemma 3.3.1 *There exists an algorithm $\mathcal{A}$ that samples edges from $M \cap E_I$ so that for every two edges e_1 and e_2 in $M \cap E_I$:*

$$\left(1 - \frac{\epsilon}{70}\right) \Pr_{e \in_R \mathcal{A}}[e = e_2] \leq \Pr_{e \in_R \mathcal{A}}[e = e_1] \leq \left(1 + \frac{\epsilon}{70}\right) \Pr_{e \in_R \mathcal{A}}[e = e_2]$$

The probability that the algorithm succeeds is at least $\frac{1}{O(\sqrt{n} \log^{1.5} n / I[f]\epsilon)}$. If it succeeds, the algorithm makes $O(\log n)$ queries, and if it fails, it makes only $O(1)$ queries. In either case, it runs in time polynomial in n.

Remark 3 *Through the standard repetition technique, the probability of error can be decreased to an arbitrarily small constant, at the cost of $O(\frac{\sqrt{n} \log^{1.5} n}{I[f]\epsilon})$ queries. Then, the run-time still stays polynomial in n, since $I[f] \geq 1/n^C$.*

Remark 4 *The distribution $\mathcal{A}$ outputs is point-wise close to the uniform distribution over $M \cap E_I$. We will also obtain such approximations to other distributions in further lemmas. Note that this requirement is stronger than closeness in L_1 norm.*

Proof: We restate the algorithm in Figure 3.3.

Algorithm $\mathcal{A}$ (given oracle access to a monotone function $f : \{0,1\}^n \to \{0,1\}$ and a parameter ϵ)

1. Assign $w = \frac{\epsilon}{3100\eta_1}\sqrt{\frac{n}{\log n}}$
2. Pick x uniformly at random from $\{0,1\}^n$.
3. Perform a descending walk P_1 downwards in the hypercube starting at x. Stop at a vertex y either after w steps, or if you hit the all-zeros vertex. Query the value of f only at the endpoints x and y of this path.
4. If $f(x) = f(y)$ output FAIL.
5. If $f(x) \neq f(y)$ perform a binary search on the path P_1 and find an influential edge e_{inf}.
6. If $e_{inf} \in M$ return e_{inf}. Otherwise output FAIL.

Figure 3-2: Algorithm for sampling an influential edge (restated)

Note that P_1 is distributed symmetrically with respect to a change of indexes. Pick an arbitrary edge e_0 in $M \cap E_I$ and let $e_0 = (v_1, v_2)$, so $v_2 \preceq v_1$. Recall that Λ_{e_0} was the set of edges in the same "layer" of the hypercube and that $|\Lambda_{e_0}|$ equals $L(v_1)\binom{n}{L(v_1)}$.

For any e' in Λ_{e_0}, e_0 and e' are different only up to a permutation of indexes. This permutation of indexes induces a bijection between the set of descending paths going through e_0 and those going through e'. Furthermore, since P_1 is distributed symmetrically with respect to a change of indexes, this bijection is between paths that have the same probability to be P_1. Therefore, the probability that P_1 passes through e' equals to the probability of it passing through e_0. Additionally, since P_1 is descending it can cross at most one edge in Λ_{e_0}, which implies that the events of P_1 crossing each of these edges are disjoint. Thus, we can write:

$$\Pr_{\mathcal{A}}[e_0 \in P_1] = \frac{\Pr_{\mathcal{A}}[P_1 \cap \Lambda_{e_0} \neq \emptyset]}{L(v_1)\binom{n}{L(v_1)}}$$

But P_1 will intersect Λ_{e_0} if and only if $L(v_1) \leq L(x) \leq L(v_1) + w - 1$. This allows us to express the probability in the numerator as a sum over the w layers of the hypercube right

above Λ_{e_0}.

$$\Pr_{\mathcal{A}}[e_0 \in P_1] = \frac{\sum_{l=L(v_1)}^{L(v_1)+w-1} \frac{1}{2^n}\binom{n}{l}}{L(v_1)\binom{n}{L(v_1)}} \tag{3.4}$$

Observation 2.1.3 allows us to lower-bound ϵ and argue that for sufficiently large n we have:

$$\left(1 - \frac{\epsilon}{1300}\right)\frac{2}{n} \leq \frac{2}{n}\frac{1}{1+2t_1/n} \leq \frac{1}{L(v_1)} \leq \frac{2}{n}\frac{1}{1-2t_1/n} \leq \left(1 + \frac{\epsilon}{1300}\right)\frac{2}{n} \tag{3.5}$$

Since $e \in M$, it is the case that $\frac{n}{2} - t_1 \leq L(v_2) \leq L(v_1) \leq \frac{n}{2} + t_1$. This allows us to use Lemma 2.1.1 and deduce that $1 - \epsilon/310 \leq \binom{n}{l}/\binom{n}{L(v_1)} \leq 1 + \epsilon/310$. This and Equation (3.5) allow us to approximate $\Pr_{\mathcal{A}}[e_0 \in P_1]$ in Equation (3.4) the following way:

$$\left(1 - \frac{\epsilon}{150}\right)\frac{w}{n2^{n-1}} \leq \Pr_{\mathcal{A}}[e_0 \in P_1] \leq \left(1 + \frac{\epsilon}{150}\right)\frac{w}{n2^{n-1}} \tag{3.6}$$

The algorithm outputs an influential edge if and only if P_1 hits an influential edge in $M \cap E_I$. At the same time, these events corresponding to different edges in $M \cap E_I$ are disjoint since P_1 can hit only at most one influential edge. This together with Bayes rule allows us to express the probability that the algorithm outputs e_0, conditioned on it succeeding:

$$\begin{aligned}\Pr_{e \in_R \mathcal{A}}[e = e_0] &= \Pr_{\mathcal{A}}[e_{inf} = e_0] \\ &= \Pr_{\mathcal{A}}\left[e_0 \in P_1 \middle| \bigvee_{e' \in M \cap E_I} (e' \in P_1)\right] = \frac{\Pr_{\mathcal{A}}\left[e_0 \in P_1\right]}{\Pr_{\mathcal{A}}\left[\bigvee_{e' \in M \cap E_I}(e' \in P_1)\right]}\end{aligned} \tag{3.7}$$

Substituting Equation (3.7) into Equation (3.6) we get:

$$\left(1 - \frac{\epsilon}{150}\right)\frac{w}{n2^{n-1}} \cdot \Pr_{\mathcal{A}}\left[\bigvee_{e_1 \in M \cap E_I} (e_1 \in P_1)\right] \leq \Pr_{e \in_R \mathcal{A}}[e = e_0] \leq \left(1 + \frac{\epsilon}{150}\right)\frac{w}{n2^{n-1}} \cdot \Pr_{\mathcal{A}}\left[\bigvee_{e_1 \in M \cap E_I} (e_1 \in P_1)\right]$$

Substituting two different edges e_1 and e_2 in $E_I \cap M$ in place of e_0 and then dividing

the resulting inequalities gives us:

$$\frac{1-\epsilon/150}{1+\epsilon/150}\Pr_{e\in_R\mathcal{A}}[e=e_2] \leq \Pr_{e\in_R\mathcal{A}}[e=e_1] \leq \frac{1+\epsilon/150}{1-\epsilon/150}\Pr_{e\in_R\mathcal{A}}[e=e_2]$$

From this, the correctness of the algorithm follows. In case of failure, it makes $O(1)$ queries and in case of success, it makes $O(\log\epsilon + \log(n/\log n)) = O(\log n)$ queries, because of the additional binary search. In either case, the run-time is polynomial.

Now, regarding the probability of success, note that the events of P_1 crossing different edges in $E_I \cap M$ are disjoint since the function is monotone. Therefore, we can sum Equation (3.6) over all edges in $M \cap E_I$ and get that the probability of success is at least $\Theta\left(\frac{|E_I\cap M|w}{n2^{n-1}}\right)$. Applying Lemma 3.1.2 and substituting w, the inverse of the success probability is:

$$O\left(\frac{n2^{n-1}}{|E_I\cap M|w}\cdot\log n\right) = O\left(\frac{n}{I[f]w}\cdot\log n\right) = O\left(\frac{\log^{1.5}(n)\sqrt{n}}{I[f]\epsilon}\right)$$

■

The following lemma, roughly speaking, shows that just as in previous lemma, the probability that P_1 in D hits an influential edge e does not depend on where exactly e is, as long as it is in $M \cap E_I$. The techniques we use are similar to the ones in the previous lemma and it follows the same outline. However here we encounter additional difficulties for two reasons: first of all, the length of P_1 is not fixed, but it is drawn from a probability distribution. Secondly, this probability distribution depends on the starting point of P_1.

Note that unlike what we have in the previous lemma, here P_1 comes from the ascending-descending view of noise sensitivity.

Lemma 3.3.2 *For any edge $e \in M \cap E_I$ it is the case that:*

$$\left(1-\frac{\epsilon}{310}\right)\frac{\delta}{2^n} \leq p_e \leq \left(1+\frac{\epsilon}{310}\right)\frac{\delta}{2^n}$$

Proof: Let $e=(v_1,v_2)$, so $v_2 \preceq v_1$. We can use the same argument from symmetry as in

the proof of Lemma 3.3.1. We get:

$$p_e = \Pr_D[(e \in P_1) \wedge \overline{E_1} \wedge \overline{E_2}] = \Pr_D[(e \in P_1) \wedge \overline{E_1}]$$
$$= \Pr_D[(e \in P_1) \wedge (L(x) - L(v_1) < t_2)] = \frac{\Pr_D[(P_1 \cap \Lambda_e \neq \emptyset) \wedge (L(x) - L(v_1) < t_2)]}{L(v_1)\binom{n}{L(v_1)}}$$

Above we did the following: Recall that E_2 is the event that P_1 crosses an edge in $E_I \cap \overline{M}$. The first equality is true because $e \in E_I \cap M$ and P_1 can cross at most one influential edge, which implies that E_2 cannot happen. For the second equality we substituted the definition of E_1. For the third equality we used the symmetry of D with respect to change of indexes just as in the proof of Lemma 3.3.1.

Recall that we defined:

$$Q_{l;w} \stackrel{\text{def}}{=} \Pr_D[\left|P_1\right| \geq w | L(x) = l]$$

This allows us to rewrite:

$$p_e = \frac{\sum_{i=1}^{t_2} \frac{1}{2^n}\binom{n}{L(v_1)+i-1} Q_{L(v_1)+i-1;i}}{L(v_1)\binom{n}{L(v_1)}} \tag{3.8}$$

Above we just looked at each of the layers of the hypercube and summed the contributions from them.

To prove a bound on p_e we will need a bound on t_2. Observation 2.1.3 implies that for any H we can assume that $\sqrt{n}\delta \leq \frac{\epsilon}{H \log^{1.5} n}$ using which we deduce

$$t_2 = n\delta(1 + 3\eta_2 \log(n)) \leq 4\eta_2 n\delta \log(n) \leq \frac{4\eta_2 \epsilon}{H}\sqrt{\frac{n}{\log n}}$$

Furthermore, we will need a bound on the binomial coefficients in Equation (3.8). By picking H to be a large enough constant, this allows us to use Lemma 2.1.1 to bound the

ratios of the binomial coefficients. For any i between 1 and t_2 inclusive:

$$1 - \frac{\epsilon}{1300} \leq \frac{\binom{n}{L(v_1)+i-1}}{\binom{n}{L(v_1)}} \leq 1 + \frac{\epsilon}{1300} \tag{3.9}$$

Equation (3.5) is valid in this setting too. Substituting Equation (3.5) together with Equation (3.9) into Equation (3.8) we get:

$$\left(1 - \frac{\epsilon}{630}\right) \frac{1}{n2^{n-1}} \sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i} \leq p_e \leq \left(1 + \frac{\epsilon}{630}\right) \frac{1}{n2^{n-1}} \sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i} \tag{3.10}$$

Observe that a vertex on the level $l+1$ has more ones than a vertex on level l. So for a positive integer w, the probability that at least w of them will flip is larger. Therefore, $Q_{l,w} \leq Q_{l+1,w}$. This allows us to bound:

$$\sum_{i=1}^{t_2} Q_{L(v_1);i} \leq \sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i} \leq \sum_{i=1}^{t_2} Q_{L(v_1)+t_2-1;i} \tag{3.11}$$

Then, using Observation 2.1.3:

$$\begin{aligned} \sum_{i=1}^{t_2} Q_{L(v_1)+t_2-1;i} \leq \sum_{i=1}^{n} Q_{L(v_1)+t_2-1;i} &= E_D[L(x) - L(y) | L(x) = L(v_1) + t_2 - 1] \\ &= \delta(L(v_1) + t_2 - 1) \leq \delta(n/2 + t_1 + t_2) \leq \left(1 + \frac{\epsilon}{630}\right) \frac{n\delta}{2} \end{aligned} \tag{3.12}$$

Now, we bound from the other side:

$$\sum_{i=1}^{t_2} Q_{L(v_1);i} = \sum_{i=1}^{n} Q_{L(v_1);i} - \sum_{i=t_2+1}^{n} Q_{L(v_1);i} \geq \sum_{i=1}^{n} Q_{L(v_1);i} - n \cdot Q_{L(v_1);t_2} \tag{3.13}$$

We bound the first term just as in Equation (3.12), and we bound the second one using a

Chernoff bound:

$$\sum_{i=1}^{n} Q_{L(v_1);i} = \delta L(v_1) \geq \delta(n/2 - t_1) \geq \left(1 - \frac{\epsilon}{1300}\right)\frac{\delta n}{2} \tag{3.14}$$

$$n \cdot Q_{L(v_1);t_2} \leq n \cdot \exp(-\frac{1}{3}n \cdot \delta 3\eta_2 \log n) \leq \frac{1}{n^{\eta_2 - 1}} \tag{3.15}$$

Substituting Equations (3.14) and (3.15) into Equation (3.13) and using Observation 2.1.3 we get:

$$\sum_{i=1}^{t_2} Q_{L(v_1);i} \geq \left(1 - \frac{\epsilon}{1300}\right)\frac{\delta n}{2} - \frac{1}{n^{\eta_2 - 1}} \geq \left(1 - \frac{\epsilon}{630}\right)\frac{\delta n}{2} \tag{3.16}$$

Substituting Equations (3.16) and (3.12) into Equation (3.11) we get:

$$\left(1 - \frac{\epsilon}{630}\right)\frac{\delta n}{2} \leq \sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i} \leq \left(1 + \frac{\epsilon}{630}\right)\frac{\delta n}{2}$$

Combining this with Equation (3.10) we deduce that:

$$\left(1 - \frac{\epsilon}{310}\right)\frac{\delta}{2^n} \leq p_e \leq \left(1 + \frac{\epsilon}{310}\right)\frac{\delta}{2^n}$$

■

3.4 Sampling descending and ascending paths going through a given influential edge.

While we will use Lemma 3.3.2 in order to estimate p_A, we will use the machinery developed in this section to estimate p_B in Section 3.5. To that end, we will need to sample from a distribution of descending and ascending paths going through a given edge. The requirement on the distribution is that it should be close to the conditional distribution of such paths P_1 that would arise from process D, conditioned on going through e and satisfying $\bar{E}_1$ and $\bar{E}_2$. See a more formal explanation in the statements of the lemmas.

In terms of resource consumption, the algorithms in this section require no queries to f but only run-time. Note that the following simple exponential time algorithm achieves

the correctness guarantee of Lemma 3.4.3 and still does not need to make any new queries to the function: The algorithm repeatedly samples P_1 and P_2 from the process D until it is the case that P_1 crosses the given edge e and neither E_1 nor E_2 happen. When this condition is satisfied the algorithm outputs these paths P_1 and P_2. The resulting distribution would exactly equal the distribution we are trying to approximate in Lemma 3.4.3, which is the ultimate goal of the section. The polynomial run-time constraint compels us to do something else. Furthermore, this is the only part of the algorithm for which the polynomial time constraint is non-trivial.

A first approach to sampling P_1 would be to take random walks in opposite directions from the endpoints of the edge e and then concatenate them together. This is in fact what we do. However, difficulty comes from determining the appropriate lengths of the walks for the following reason. If P_1 is longer, it is more likely to hit the influential edge e. This biases the distribution of the descending paths hitting e towards the longer descending paths. In order to accommodate for this fact we used the following two-step approach:

1. Sample only the levels of the starting and ending points of the path P_1. This is equivalent to sampling the length of the segment of P_1 before the edge e and after it. This requires careful use of rejection sampling together with the techniques we used to prove Lemmas 3.3.1 and 3.3.2. Roughly speaking, we use the fact that P_1 is distributed symmetrically with respect to the change of indices in order to reduce a question about the edge e to a question about the layer Λ_e. Then, we use the Lemma 2.1.1 to answer questions about random walks hitting a given layer. This is handled in Lemma 3.4.1.

2. Sample a path P_1 that has the given starting and ending levels and passes through an influential edge e. This part is relatively straightforward. We prove that all the paths satisfying these criteria are equally likely. We sample one of them randomly by performing two random walks in opposite directions starting at the endpoints of e. This all is handled in Lemma 3.4.3.

Lemma 3.4.1 *There is an algorithm $\mathcal{W}$ that takes as input an edge $e = (v_1, v_2)$ in $M \cap E_I$, so that $v_2 \preceq v_1$, and samples two non-negative numbers w_1 and w_2, so that for any two*

Algorithm $\mathcal{W}$ (given an edge $e \stackrel{\text{def}}{=} (v_1, v_2)$ so $v_2 \preceq v_1$)

1. Pick an integer l uniformly at random among the integers in $[L(v_1), L(v_1) + t_2 - 1]$. Pick a vertex x randomly at level l.
2. As in phase 1 of the noise sensitivity process, traverse in random order through the indices of x and for each index that equals to one, flip it with probability δ. The intermediate results form a path P_1, and we call its endpoint y.
3. If P_1 does not intersect Λ_e go to step 1.
4. Otherwise, output $w_1 = L(x) - L(v_1)$ and $w_2 = L(v_2) - L(y)$.

Figure 3-3: Algorithm $\mathcal{W}$ (restated)

non-negative w_1' and w_2':

$$\left(1 - \frac{\epsilon}{70}\right) \Pr_{\mathcal{W}(e)}[(w_1 = w_1') \wedge (w_2 = w_2')]$$
$$\leq \Pr_D[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') | (e \in P_1) \wedge \overline{E_1} \wedge \overline{E_2}]$$
$$\leq \left(1 + \frac{\epsilon}{70}\right) \Pr_{\mathcal{W}(e)}[(w_1 = w_1') \wedge (w_2 = w_2')] \quad (3.17)$$

The algorithm requires no queries to f and runs in time polynomial in n.

Remark 5 *the approximation guarantee here is similar to the one in Lemma 3.3.1. It guaranteed that the relative distance should be small point-wise. This guarantee is stronger than closeness in either L_1 and L_∞ norms. We also employ an analogous approximation guarantee in Lemma 3.4.3.*

Proof: We restate Algorithm $\mathcal{W}$ in Figure 3.4 (recall that we used the symbol Λ_e to denote the whole layer of edges that have the same endpoint levels as e).

To prove the correctness of the algorithm, we begin by simplifying the expression above. If $e \in P_1$, then P_1 cannot contain any other influential edge, so E_2 cannot happen and we can drop it from notation. Additionally we can substitute the definition for E_1

so:

$$\Pr_D[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') | e \in P_1 \wedge \overline{E_1} \wedge \overline{E_2}]$$
$$= \Pr_D \left[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') \middle| e \in P_1 \wedge (L(x) - L(v_1) < t_2) \right]$$

Now we can use the fact that D is symmetric with respect to the change of indices, so we can substitute e with any e' in Λ_e. Therefore:

$$\Pr_D[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') | (e \in P_1) \wedge \overline{E_1} \wedge \overline{E_2}]$$
$$= \frac{1}{|\Lambda_e|} \cdot \sum_{e' \in \Lambda_e} \Pr_D \Bigg[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') \Bigg|$$
$$e' \in P_1 \wedge (L(x) - L(v_1) < t_2) \Bigg]$$
$$= \sum_{e' \in \Lambda_e} \Pr_D \left[L(x) - L(v_1) = w_1' \wedge (L(v_2) - L(y) = w_2') \middle| e' \in P_1 \wedge (L(x) - L(v_1) < t_2) \right]$$
$$\times \Pr_D \left[e' \in P_1 \middle| P_1 \cap \Lambda_e \neq \emptyset \right]$$
$$= \Pr_D \Bigg[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') \Bigg|$$
$$(P_1 \cap \Lambda_e \neq \emptyset) \wedge (L(x) - L(v_1) < t_2) \Bigg] \quad (3.18)$$

Observation 3.4.2 *The algorithm never returns $w_1 \geq t_2$, which is appropriate since the distribution being approximated is conditioned on $L(x) - L(v_1) < t_2$. In what follows we consider $1 \leq w_1 < t_2$ and $1 \leq w_1' < t_2$. Additionally, recall that w_2' is non-negative.*

Before we continue, we derive the following intermediate results. Because of how P_1 is chosen in $\mathcal{W}$, we can use the notation:

$$Q_{l;w} = \Pr_D[\left|P_1\right| \geq w | L(x) = l] = \Pr_{\mathcal{W}(e)}[\left|P_1\right| \geq w | L(x) = l]$$

The last equality is true because after picking x both process D and algorithm $\mathcal{W}(e)$ pick the path P_1 exactly the same way. Therefore, the path P_1 is distributed the same way for

the both processes if we condition on them picking the same starting point x.

We can express:

$$\begin{aligned}
&\Pr_{\mathcal{W}(e)}[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2')] = \\
&\qquad \frac{1}{t_2}\Pr_{\mathcal{W}(e)}[L(v_2) - L(y) = w_2' | L(x) - L(v_1) = w_1'] \\
&\qquad = \frac{1}{t_2}\Pr_{D}[L(v_2) - L(y) = w_2' | L(x) - L(v_1) = w_1'] \\
&= \frac{1}{t_2} Q_{L(v_1)+w_1';w_1'+1} \Pr_D \left[L(v_2) - L(y) = w_2' \middle| (L(x) - L(v_1) = w_1') \wedge (|P_1| \geq w_1' + 1) \right]
\end{aligned} \tag{3.19}$$

Above: (i) The first equality uses the fact that in $\mathcal{W}(e)$ the level of x is chosen uniformly between the t_2 levels, and therefore $\Pr_{\mathcal{W}(e)}[L(x) - L(v_1) = w_1'] = 1/t_2$. (ii) The second inequality uses the fact that after picking x, both $W(e)$ and D pick P_1 and consequently y identically. (iii) The third inequality is an application of the law of conditional probabilities together with the definition of $Q_{l,w}$.

Building on the previous equality we have:

$$\begin{aligned}
&\Pr_{\mathcal{W}(e)}[(w_1 = w_1') \wedge (w_2 = w_2')] = \\
&\qquad \Pr_{\mathcal{W}(e)} \left[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') \middle| P_1 \cap \Lambda_e \neq \emptyset \right] \\
&\qquad = \frac{\Pr_{\mathcal{W}(e)}[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2') \wedge (P_1 \cap \Lambda_e \neq \emptyset)]}{\Pr_{\mathcal{W}}[P_1 \cap \Lambda_e \neq \emptyset]]} \\
&\qquad = \frac{\Pr_{\mathcal{W}(e)}[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2')]}{\Pr_{\mathcal{W}}[P_1 \cap \Lambda_e \neq \emptyset]]} \\
&= \frac{Q_{L(v_1)+w_1';w_1'+1} \Pr_D \left[L(v_2) - L(y) = w_2' \middle| (L(x) = L(v_1) + w_1') \wedge (|P_1| \geq w_1' + 1) \right]}{\sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i}}
\end{aligned} \tag{3.20}$$

Above: (i) At step (5) of the algorithm, we have $w_1 = L(x) - L(v_1)$ and $w_2 = L(v_2) - L(y)$,

but this happens only after the condition on step (4) is satisfied. This adds a conditioning at the first equality. (ii) The second equality comes from Observation 3.4.2 since $w_1' \geq 1$ and $w_2' \geq 0$, hence Λ_e starts above Λ_e and ends below it. Thus, the first two clauses imply the last one, so we drop it. (iii) Regarding the third equality, in the numerator we substituted Equation (3.19) whereas in the denominator we computed $\Pr_{\mathcal{W}}[P_1 \cap \Lambda_e \neq \emptyset]$ by breaking it into contributions from the t_2 levels above Λ_e. Finally, we canceled $\frac{1}{t_2}$ from both the numerator and the denominator.

Back to Equation (3.18). Analogous to how we derived Equation (3.19), we have:

$$\begin{aligned}
&\Pr_D[(L(x) - L(v_1) = w_1') \wedge (L(v_2) - L(y) = w_2')] \\
&\quad = \Pr_D[L(x) - L(v_1) = w_1'] \cdot \Pr_D\left[L(v_2) - L(y) = w_2' \middle| L(x) - L(v_1) = w_1'\right] \\
&\quad = \frac{1}{2^n}\binom{n}{L(v_1) + w_1'} Q_{L(v_1)+w_1';w_1'+1} \cdot \Pr_D\Bigg[L(v_2) - L(y) = w_2' \Bigg| \\
&\qquad\qquad (L(x) - L(v_1) = w_1') \wedge (|P_1| \geq w_1' + 1)\Bigg]
\end{aligned}$$

In addition to the steps analogous to the ones involved in getting Equation (3.18), above we used the fact that the layer $L(v_1) + w_1'$ has $\binom{n}{L(v_1)+w_1'} Q_{L(v_1)+w_1';w_1'+1}$ vertices out of the 2^n vertices overall.

Again, the same way we derived Equation (3.20) from Equation (3.19) using Bayes rule, we get:

$$
\begin{aligned}
&\Pr_D\Bigg[(L(x)-L(v_1)=w_1')\wedge(L(v_2)-L(y)=w_2')\Bigg| \\
&\qquad\qquad (P_1\cap\Lambda_e\neq\emptyset)\wedge(L(x)-L(v_1)<t_2)\Bigg] \\
&= \frac{1}{\Pr_D[(P_1\cap\Lambda_e\neq\emptyset)\wedge(L(x)-L(v_1)<t_2)]}\cdot\Pr_D\Bigg[(L(x)-L(v_1)=w_1')\wedge \\
&\qquad (L(v_2)-L(y)=w_2')\wedge(P_1\cap\Lambda_e\neq\emptyset)\wedge(L(x)-L(v_1)<t_2)\Bigg] \\
&\qquad = \frac{\Pr_D[(L(x)-L(v_1)=w_1')\wedge(L(v_2)-L(y)=w_2')]}{\Pr_D[(P_1\cap\Lambda_e\neq\emptyset)\wedge(L(x)-L(v_1)<t_2)]} \\
&\quad = \frac{1}{\sum_{i=1}^{t_2}\frac{1}{2^n}\binom{n}{L(v_1)+i-1}Q_{L(v_1)+i-1;i}}\cdot\frac{1}{2^n}\binom{n}{L(v_1)+w_1'}Q_{L(v_1)+w_1';w_1'+1} \\
&\quad \times\Pr_D\Bigg[L(v_2)-L(y)=w_2'\Bigg|(L(x)-L(v_1)=w_1')\wedge(|P_1|\geq w_1'+1)\Bigg] \qquad (3.21)
\end{aligned}
$$

In the second equality we dropped the clauses $(P_1\cap\Lambda_e\neq\emptyset)$ and $(L(x)-L(v_1)<t_2)$ because by Observation 3.4.2 they are implied by the first two clauses.

Since by Observation 3.4.2 it is the case that $1\leq i\leq t_2$ and $0\leq w_1'<t_2$, the same way we proved Equation (3.9) we have:

$$
1-\frac{\epsilon}{150}\leq\frac{\binom{n}{L(v_1)+i-1}}{\binom{n}{L(v_1)+w_1'}}\leq 1+\frac{\epsilon}{150} \qquad (3.22)
$$

Combining Equations (3.21) and (3.22) we get:

$$
\begin{aligned}
&\left(1-\frac{\epsilon}{70}\right)\frac{Q_{L(v_1)+w_1';w_1'+1}}{\sum_{i=1}^{t_2}Q_{L(v_1)+i-1;i}} \\
&\qquad\times\Pr_D[L(v_2)-L(y)=w_2'|(L(x)-L(v_1)=w_1')\wedge(|P_1|\geq w_1'+1)] \\
&\leq\Pr_D[(L(x)-L(v_1)=w_1')\wedge(L(v_2)-L(y)=w_2')|(P_1\cap\Lambda_e\neq\emptyset)\wedge(L(x)-L(v_1)\leq t_2)] \\
&\qquad\leq\left(1+\frac{\epsilon}{70}\right)\frac{Q_{L(v_1)+w_1';w_1'+1}}{\sum_{i=1}^{t_2}Q_{L(v_1)+i-1;i}} \\
&\qquad\times\Pr_D[L(v_2)-L(y)=w_2'|(L(x)-L(v_1)=w_1')\wedge(|P_1|\geq w_1'+1)] \qquad (3.23)
\end{aligned}
$$

Substituting Equations (3.18) and (3.20) into Equation (3.23) proves the correctness of the algorithm.

Now, we consider the run-time. Since $L(x) \in [L(v_1), L(v_1) + t_2 - 1]$ and $L(v_1) \geq n/2 - t_1 \geq n/4$, then $L(x) \geq n/4$. In expectation at least $\delta n/4$ of these indices, which equal to one, should flip. We can use a Chernoff bound to bound the probability of less than $\delta n/8$ of them flipping by $\exp\left(-\frac{1/4 \cdot n\delta/4}{2}\right) = \exp\left(-\frac{n\delta}{32}\right) \leq \exp(-\frac{1}{32})$. Therefore:

$$\Pr_{\mathcal{W}}[|P_1| \geq n\delta/8] \geq \Theta(1)$$

If this happens, it is sufficient for l to be less than $L(v_1) + n\delta/8$ for P_1 to intersect Λ_e. The probability of this happening is at least $\frac{n\delta/8}{t_2}$. Therefore, we can conclude:

$$\Pr_{\mathcal{W}}[P_1 \cap \Lambda_e \neq \emptyset] \geq \Omega\left(\frac{n\delta}{t_2}\right)$$

Then, the number of time the algorithm goes through the loop is $O(t_2/(n\delta)) = \tilde{O}(1)$. Thus, the algorithm runs in polynomial time. ■

Lemma 3.4.3 *There exists an algorithm $\mathcal{B}$ with the following properties. It takes as input an edge $e = (v_1, v_2)$ in $M \cap E_I$, so that $v_2 \preceq v_1$ and outputs paths P_1 and P_2 together with hypercube vertices x, y and z. It is the case that x is the starting vertex of P_1, y is both the starting vertex of P_2 and the last vertex of P_1, and z is the last vertex of P_2. Additionally, P_1 is descending and P_2 is ascending. Furthermore, for any pair of paths P_1' and P_2' we have:*

$$\left|\Pr_{\mathcal{B}(e)}[(P_1 = P_1') \wedge (P_2 = P_2')] - \Pr_D[(P_1 = P_1') \wedge (P_2 = P_2') | (e \in P_1) \wedge \overline{E_1} \wedge \overline{E_2}]\right| \leq \frac{\epsilon}{70} \Pr_{\mathcal{B}(e)}[(P_1 = P_1') \wedge (P_2 = P_2')] \quad (3.24)$$

It requires no queries to the function and takes computation time polynomial in n to draw one sample.

Proof: Below we restate the algorithm in Figure 3.4. Now, we analyze the algorithm.

Algorithm $\mathcal{B}$ (given an influential edge $e \stackrel{\text{def}}{=} (v_1, v_2)$ so $v_2 \preceq v_1$)

1. Use $\mathcal{W}(e)$ to sample w_1 and w_2.
2. Perform an ascending random walk of length w_1 starting at v_1 and call its endpoint x. Similarly, perform a descending random walk starting at v_2 of length w_2, call its endpoint y.
3. Define P_1 as the descending path that results between x and y by concatenating the two paths from above, oriented appropriately, and edge e.
4. Define P_2 just as in phase 2 of our process starting at y. Consider in random order all the zero indices y has in common with x and flip each with probability δ.
5. Return P_1 ,P_2, x, y and z.

Figure 3-4: Algorithm $\mathcal{B}$ (restated)

Without loss of generality we assume that P_1' is descending and starts at a vertex x' and ends at a vertex y'. P_2', in turn, is ascending, starts at y' and ends at z'. For all the other paths all the probabilities in (24) equal to zero. We have that:

$$\begin{aligned}
&\Pr_D\left[(P_1 = P_1') \wedge (P_2 = P_2') \middle| (e \in P_1) \wedge \overline{E_1} \wedge \overline{E_2}\right] \\
&\qquad = \Pr_D\left[(L(x) = L(x')) \wedge (L(y) = L(y')) \middle| (e \in P_1) \wedge \overline{E_1} \wedge \overline{E_2}\right] \\
&\times \Pr_D\left[(P_1 = P_1') \wedge (P_2 = P_2') \middle| (e \in P_1) \wedge \overline{E_1} \wedge \overline{E_2} \wedge (L(x) = L(x')) \wedge (L(y) = L(y'))\right]
\end{aligned} \tag{3.25}$$

Similarly, we can write:

$$\begin{aligned}
\Pr_{\mathcal{B}(e)}[(P_1 = P_1') \wedge (P_2 = P_2')] &= \Pr_{\mathcal{B}(e)}[(L(x) = L(x')) \wedge (L(y) = L(y'))] \\
&\times \Pr_{\mathcal{B}(e)}\left[(P_1 = P_1') \wedge (P_2 = P_2') \middle| (L(x) = L(x')) \wedge (L(y) = L(y'))\right]
\end{aligned} \tag{3.26}$$

By Lemma 3.4.1 we know that:

$$\left(1-\frac{\epsilon}{70}\right)\Pr_{\mathcal{B}(e)}[(L(x)=L(x'))\wedge(L(y)=L(y'))]$$
$$\leq \Pr_D\left[(L(x)=L(x'))\wedge(L(y)=L(y'))\middle|(e\in P_1)\wedge\overline{E_1}\wedge\overline{E_2}\right]$$
$$\leq \left(1+\frac{\epsilon}{70}\right)\Pr_{\mathcal{B}(e)}[(L(x)=L(x'))\wedge(L(y)=L(y'))] \quad (3.27)$$

Considering Equations (3.25), (3.26) and (3.27) together, for the lemma to be true, it is enough that:

$$\Pr_D[(P_1=P_1')\wedge(P_2=P_2')|e\in P_1\wedge\overline{E_1}\wedge\overline{E_2}\wedge(L(x)=L(x'))\wedge(L(y)=L(y'))]$$
$$=\Pr_{\mathcal{B}(e)}[(P_1=P_1')\wedge(P_2=P_2')|(L(x)=L(x'))\wedge(L(y)=L(y'))] \quad (3.28)$$

Since e is in $M\cap E_I$, if $e\in P_1$ then E_2 cannot happen. If $L(x')\geq L(v_1)+t_2$ or $L(x')<L(v_1)$ both sides are zero. Otherwise, E_1 cannot happen either and Equation (3.28) is equivalent to:

$$\Pr_D[(P_1=P_1')\wedge(P_2=P_2')|(e\in P_1)\wedge(L(x)=L(x'))\wedge(L(y)=L(y'))]$$
$$=\Pr_{\mathcal{B}(e)}[(P_1=P_1')\wedge(P_2=P_2')|(L(x)=L(x'))\wedge(L(y)=L(y'))] \quad (3.29)$$

We first argue that:

$$\Pr_D[P_1=P_1'|(e\in P_1)\wedge(L(x)=L(x'))\wedge(L(y)=L(y'))]=$$
$$\Pr_{\mathcal{B}(e)}[P_1=P_1'|(L(x)=L(x'))\wedge(L(y)=L(y'))] \quad (3.30)$$

This comes from the fact that in both D and $\mathcal{B}(e)$ the distribution of P_1 is symmetric with

respect to exchange of coordinates. Now we argue that:

$$\begin{aligned}\Pr_D[P_2 = P_2' | (e \in P_1) \wedge (L(x) = L(x')) \wedge (L(y) = L(y')) \wedge (P_1 = P_1')] \\ = \Pr_{\mathcal{B}(e)}[P_2 = P_2' | (L(x) = L(x')) \wedge (L(y) = L(y')) \wedge (P_1 = P_1')] \quad (3.31)\end{aligned}$$

This is true because given P_1, both D and $\mathcal{B}(e)$ choose P_2 the same way. Combining Equation (3.31) with Equation (3.30) we get Equation (3.29) which completes the proof of correctness.

The claims about run-time follow from the run-time guarantee on $\mathcal{W}$. ■

Algorithm for estimating noise sensitivity. (given oracle access to a monotone function $f\colon \{0,1\}^n \to \{0,1\}$, and a parameter ϵ)

1. Using the algorithm from [25] as described in Theorem 5, compute an approximation to the influence of f to within a multiplicative factor of $(1 \pm \epsilon/33)$. This gives us $\tilde{I}$.
2. Compute $\tilde{p}_A := \delta\tilde{I}/2$.
3. Initialize $\alpha := 0$ and $\beta := 0$. Repeat the following until $\alpha = \frac{768 \ln 200}{\epsilon^2}$.
   - Use algorithm $\mathcal{A}$ from Lemma 3.3.1 repeatedly to successfully sample an edge e.
   - From Lemma 3.4.3 use the algorithm $\mathcal{B}$, giving it e as input, and sample P_1, P_2, x, y and z.
   - If it is the case that $f(x) = 1$ and $f(z) = 0$, then $\alpha := \alpha + 1$.
   - $\beta := \beta + 1$.
4. Set $\tilde{p}_B = \frac{\alpha}{\beta}$.
5. Return $2\tilde{p}_A\tilde{p}_B$.

Figure 3-5: Algorithm for estimating noise sensitivity (restated).

3.5 The noise sensitivity estimation algorithm.

We restate our algorithm in Figure 3.5.

We analyze the algorithm by combining all the lemmas from the previous section. First, we prove that:

Lemma 3.5.1 *It is the case that:*

$$\left(1 - \frac{\epsilon}{150}\right)\delta I[f]/2 \leq p_A \leq \left(1 + \frac{\epsilon}{150}\right)\delta I[f]/2$$

Proof: Summing Lemma 3.3.2 over all the edges in $E_I \cap M$ we get that:

$$\left(1 - \frac{\epsilon}{310}\right)\frac{\delta \cdot |E_I \cap M|}{2^n} \leq \sum_{e \in E_I \cap M} p_e \leq \left(1 + \frac{\epsilon}{310}\right)\frac{\delta \cdot |E_I \cap M|}{2^n}$$

Substituting Equation (3.2) we get:

$$\left(1-\frac{\epsilon}{310}\right)\frac{\delta I|E_I\cap M|}{2|E_I|}\leq p_A\leq\left(1+\frac{\epsilon}{310}\right)\frac{\delta I|E_I\cap M|}{2|E_I|}$$

With Observation 3.1.2 this implies the lemma.

■

Now, we proceed to prove that $\tilde{p}_A$ and $\tilde{p}_B$ are good approximations for p_A and p_B respectively.

Corollary 3.5.2 *With probability at least* 0.99*:*

$$\left(1-\frac{\epsilon}{16}\right)\tilde{p}_A\leq p_A\leq\left(1+\frac{\epsilon}{16}\right)\tilde{p}_A$$

Proof: From the correctness of the influence estimation algorithm we have that with probability at least 0.99:

$$\left(1-\frac{\epsilon}{33}\right)\tilde{I}\leq I[f]\leq\left(1+\frac{\epsilon}{33}\right)\tilde{I}$$

Which together with our lemma implies that:

$$\left(1-\frac{\epsilon}{16}\right)\frac{\delta\tilde{I}}{2}\leq p_A\leq\left(1+\frac{\epsilon}{16}\right)\frac{\delta\tilde{I}}{2}$$

■

Definition 10 *We call an iteration of the main loop* ***successful*** *if* $f(x)=1$ *and* $f(z)=0$*. We also denote by* ϕ *the probability of any given iteration to be successful.*

In the following two lemmas, we show that $\tilde{p}_B$ is a good approximation to ϕ and that ϕ, in turn, is a good approximation to p_B. This will allow us to conclude that $\tilde{p}_B$ is a good approximation to p_B.

Lemma 3.5.3 *With probability at least* 0.99 *we have that:*

$$\left(1-\frac{\epsilon}{16}\right)\tilde{p}_B\leq\phi\leq\left(1+\frac{\epsilon}{16}\right)\tilde{p}_B$$

Additionally, the expected number of iterations of the main loop is $O(1/(\phi\epsilon^2))$.

Proof: See Appendix C, Section C.4. ■

Lemma 3.5.4 *It is the case that:*

$$\left(1-\frac{\epsilon}{16}\right)\phi \leq p_B \leq \left(1+\frac{\epsilon}{16}\right)\phi$$

Proof:

See Appendix C, Section C.5. ■

Corollary 3.5.2 with Lemmas 3.5.3 and 3.5.4 together imply that with probability at least $2/3$:

$$\left(1-\frac{\epsilon}{5}\right)\tilde{p}_A\tilde{p}_B \leq p_A p_B \leq \left(1+\frac{\epsilon}{5}\right)\tilde{p}_A\tilde{p}_B$$

Combining this with Lemma 3.1.3 and Equation (3.3) we get that:

$$\left(1-\frac{\epsilon}{2}\right)2\tilde{p}_A\tilde{p}_B \leq NS_\delta[f] \leq \left(1+\frac{\epsilon}{2}\right)2\tilde{p}_A\tilde{p}_B$$

This proves the correctness of the algorithm. Now consider the number of queries:

- Estimating the influence requires $O\left(\frac{\sqrt{n}\log(n/\epsilon)}{I[f]\epsilon^3}\right)$ queries and polynomial time. Since by Observation 2.1.3 it is the case that $\epsilon \geq 1/n$, this is at most $O\left(\frac{\sqrt{n}\log(n)}{I[f]\epsilon^3}\right)$.
- By Lemma 3.3.1, successfully sampling an edge requires $O\left(\frac{\sqrt{n}\log^{1.5} n}{I[f]\epsilon}\right)$ queries and polynomial time. By Lemma 3.4.3 for each edge we willl additionally spend a polynomial amount of extra time.
- By Lemmas 3.5.3 and 3.5.4 we will have $O\left(\frac{1}{\phi\epsilon^2}\right) = O\left(\frac{1}{p_B\epsilon^2}\right)$ iterations of the loop.

Therefore, the overall run-time is polynomial, and the overall number of queries made is:

$$O\left(\frac{\sqrt{n}\log n}{I[f]\epsilon^3}\right) + O\left(\frac{\sqrt{n}\log^{1.5} n}{I[f]\epsilon}\right)O\left(\frac{1}{p_B\epsilon^2}\right) = O\left(\frac{\sqrt{n}\log^{1.5} n}{I[f]\epsilon}\cdot\frac{1}{p_B\epsilon^2}\right)$$

Finally, using Lemmas 3.5.1 and 3.1.3 together with Equation (3.3) we get the desired bound:

$$O\left(\frac{\sqrt{n}\delta\log^{1.5}n}{p_A\epsilon}\cdot\frac{1}{p_B\epsilon^2}\right)=O\left(\frac{\sqrt{n}\delta\log^{1.5}n}{NS_\delta[f]\epsilon^3}\right)$$

Chapter 4

Lower bounding the query complexity.

We begin our proof of Theorems 2, 3 and 4 by first defining the distributions D_1^B, D_1^I and D_1^δ to consist of a single function f_0: $\{0,1\}^n \to \{0,1\}$:

$$f_0(x) = \begin{cases} 1 & \text{if } L(x) > n/2 + k\sqrt{n \log n} \\ 0 & \text{otherwise} \end{cases}$$

And here k is chosen so that $n/2 + k\sqrt{n \log n}$ is the smallest integer larger than $n/2$ for which $B[f_0] \leq 1/n^{C_1}$.

For our lower bounds we will need to show that f_0 and ORs of f_0 with other functions have useful properties. For this we will need the following lemma. Informally, it says that $B[F]$, $I[F]$ and $NS_\delta[F]$ are continuous functions of F in the sense that changing F a little bit does not change them drastically.

Lemma 4.0.1 *For any monotone function F: $\{0,1\}^n \to \{0,1\}$, denote by F' the OR of F and f_0. Then:*

a)

$$B[F] \leq B[F'] \leq B[F'] + \frac{1}{n^{C_1}}$$

b)

$$\left| I[F] - I[F'] \right| \leq \frac{2n}{n^{C_1}}$$

c) For any δ:

$$\left| NS_\delta[F] - NS_\delta[F'] \right| \leq \frac{2}{n^{C_1}}$$

Proof: See Appendix D, Section D.1. ■

Lemma 4.0.2 *It is the case that:*

a) $k \leq \sqrt{\frac{C_1}{8}}$*, and hence* k *is also a constant.*

b) $B[f_0] = 1/\Theta(n^{C_1})$

c) $\Omega(1/n^{C_1}) \leq I[f_0] \leq O(n/n^{C_1})$

e) For any δ*, it is the case that* $\Omega(1/n^{C_1+1}) \leq NS_\delta[f_0] \leq O(1/n^{C_1})$.

Proof: See Appendix C, Section D.2. ■

We will use the two following main lemmas, that we will prove in two separate subsections. The first one is a computational lemma that says that any function family that is "thin" at the level $L(x) = n/2 + k\sqrt{n \log n}$ can be transformed into two distributions that are hard to distinguish. By "thin" we mean that they have few positive points at the level right below the threshold of f_0. The second lemma says that there exist function families that are both "thin" in the sense the first lemma requires and have a large amount of bias, influence and noise sensitivity.

Lemma 4.0.3 *Suppose* $F\colon \{0,1\}^n \to \{0,1\}$ *is a monotone Boolean function with the property that:*

$$\Pr_{x \in_R \{0,1\}^n}[F(x) = 1 | L(x) = n/2 + k\sqrt{n \log n}] \leq 1/q_0$$

Additionally, suppose $\mathcal{C}$ *is an algorithm that makes* $o(q_0)$ *queries given access to the following random function:*

- *With probability* $1/2$ *it is drawn from* D_1 *that consists of only* f_0.
- *With probability* $1/2$ *it is drawn from* D_2 *that consists of an OR of the following:*

– *The function* f_0.

– $F(\sigma(x))$, *where* σ *is a random permutation of indices.*

Consequently, suppose that $\mathcal{C}$ *outputs a guess whether its input was from* D_1 *or* D_2. *Then* $\mathcal{C}$ *has to err with probability more than* $1/3$.

Lemma 4.0.4 *There exist functions* F^B: $\{0,1\}^n \to \{0,1\}$, F^I: $\{0,1\}^n \to \{0,1\}$ *and for every* $1/n \leq \delta \leq 1/2$ *there exists* F^δ: $\{0,1\}^n \to \{0,1\}$ *such that:*

- *Any* f *in* $\{F^B, F^I\} \cup \{F^\delta : 1/n \leq \delta \leq 1/2\}$ *has the property that:*

$$\Pr_{x \in_R \{0,1\}^n}[f(x) = 1 | L(x) = n/2 + k\sqrt{n \log n}] \leq \Theta\left(1/n^{C_2} \cdot e^{\sqrt{C_1 \log n/2}}\right)$$

- $B[F^B] \geq \Omega(1/n^{C_2})$.
- $I[F^I] \geq \Omega(\sqrt{n}/n^{C_2})$.
- *For any* $1/n \leq \delta \leq 1/2$, *it is the case that:*

$$NS_\delta[F^\delta] \geq \begin{cases} \Omega(\delta\sqrt{n}/n^{C_2}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Omega(1/n^{C_2}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases}$$

Recall that we defined D_1^B, D_1^I and D_1^δ to be composed of only the function f_0. Now we also have functions that are thin and have large bias, influence and noise sensitivity. Therefore, we use them to define distributions we can use with Lemma 4.0.3:

- D_2^B as the OR of $f_0(x)$ and $F^B(\sigma(x))$. Where, recall, σ is a random permutation of indices.
- D_2^I as the OR of $f_0(x)$ and $F^I(\sigma(x))$.
- For each $1/n \leq \delta \leq 1/2$, we define D_2^δ as the OR of $f_0(x)$ and $F^\delta(\sigma(x))$.

Observation 4.0.5 *Permuting the indices to an input of a function preserves its bias, influence and noise sensitivity. That, together with Lemma 4.0.1 and Lemma 4.0.4, implies that:*

a) For any f in D_2^B, we have $B[f] \geq \Omega(1/n^{C_2})$.

b) For any f in D_2^I, we have $I[f] \geq \Omega(\sqrt{n}/n^{C_2}) - O(n/n^{C_1}) = \Omega(\sqrt{n}/n^{C_2})$. The last equality is true because $C_1 - 1 > C_2$.

c) For all $1/n \leq \delta \leq 1/2$ and for all f in D_2^δ:

$$NS_\delta[f] \geq \begin{cases} \Omega(\delta\sqrt{n}/n^{C_2}) - O(1/n^{C_1}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Omega(1/n^{C_2}) - O(1/n^{C_1}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases} =$$

$$\begin{cases} \Omega(\delta\sqrt{n}/n^{C_2}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Omega(1/n^{C_2}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases}$$

Here, again, the last equality is true because $C_1 - 1 > C_2$

Now, we are ready to prove our main theorems. Let us first consider the case of estimating the bias. We will prove it by contradiction, showing that the negation of Theorem 3 implies we can reliably distinguish two distributions that Lemma 4.0.3 prevents us from distinguishing. Suppose $\mathcal{L}_B$ is an algorithm, taking as input a monotone function $f : \{0,1\}^n \to \{0,1\}$ and with probability at least $2/3$ distinguishing whether f (i) has a bias of $\Theta(1/n^{C_1})$ or (ii) has a bias of at least $\Omega(1/n^{C_2})$. For the sake of contradiction, assume that it makes $o\left(\frac{n^{C_2}}{e^{\sqrt{C_1 \log n/2}}}\right)$ queries.

By Lemma 4.0.2 every function in D_1 has a bias of $\Theta(1/n^{C_1})$ and by Observation 4.0.5, the bias of every function in D_2^B is at least $\Omega(1/n^{C_2})$. Therefore, $\mathcal{L}_B$ can distinguish between them with probability at least $2/3$ making $o\left(\frac{n^{C_2}}{e^{\sqrt{C_1 \log n/2}}}\right)$. But by Lemma 4.0.3, such an algorithm has to err with probability more than $1/3$. We have a contradiction and Theorem 3 follows.

Theorems 4 and 2 follow analogously.

4.1 Proof of Lemma 4.0.3

Suppose $\mathcal{C}$ is an adaptive algorithm that makes at most q queries and distinguishes a random function in D_2 from D_1. We denote the number of queries it makes as q. Without loss of generality, we assume that it always makes precisely q queries. The algorithm is adaptive and in the end it outputs 1 or 2. Then, the probability that the algorithm correctly distinguishes equals:

$$p_{\mathcal{C}} \stackrel{\text{def}}{=} \frac{1}{2}\Pr[\mathcal{C} \text{ returns } 1 | f \in_R D_1] + \frac{1}{2}\Pr[\mathcal{C} \text{ returns } 2 | f \in_R D_2]$$

We call the difference above **the distinguishing power** of $\mathcal{C}$.

Observation 4.1.1 *For any f in $D_1 \cup D_2$ it is the case that if $L(x) > n/2 + k\sqrt{n \log n}$, then $f(x) = 1$. Therefore, without loss of generality we can assume that $\mathcal{C}$ never queries any point in that region.*

Observation 4.1.2 *If $\mathcal{C}$ is randomized, we can think of it as a probability distribution over deterministic algorithms. The distinguishing powers of $\mathcal{C}$ then will be the weighted sum of the distinguishing power of the deterministic algorithms, weighted by their probabilities. Therefore, the distinguishing power of the best of these deterministic algorithms is at least that of $\mathcal{C}$. Thus, without loss of generality we can assume that $\mathcal{C}$ is deterministic.*

Now, since $\mathcal{C}$ is deterministic and it makes q queries, it can be represented as a decision tree of depth q. At each node, $\mathcal{C}$ queries a point and proceeds to the next node. In the end, after q queries, the algorithm reaches a leaf and outputs the label of the leaf, namely 1 or 2. We can divide this decision tree into two regions:

- A path A that the algorithm takes if at every query it receives zero.
- The rest of the decision tree. We call this region B.

By Observation 4.1.1 and the definition of f_0, when the algorithm is given access to a member of D_1, namely f_0, all the points x that it queries will have $f(x) = f_0(x) = 0$.

Therefore, the algorithm will follow the path A on the decision tree and end up on the single leaf there.

Suppose now the algorithm is given access to a function in D_2. There are two cases:

- All the queries x^j it makes, will have $f(x^j) = 0$. Then, on the decision tree it will follow the path A and end up at the single leaf on it.

- After making query x^j for which $f(x^j) = 1$, the algorithm ends up in the subset of the tree we call B. We call the probability that this happens p_{get1}.

If the single leaf in A is not labeled with 1, then the algorithm will always err, given access to D_1. Similarly, the algorithm can reach a leaf in B only if it was given access to D_2. Thus, labeling all such leaves with 2 can only increase the distinguishing power. Therefore, without loss of generality, we assume that this is the labeling used in $\mathcal{C}$. Then, the distinguishing power $p_{\mathcal{C}}$ equals $\frac{1}{2}(1 + p_{\text{get1}})$.

The following lemma shows that we can assume that the algorithm is non-adaptive and only makes queries on the level $n/2 + k\sqrt{n \log n}$.

Lemma 4.1.3 *There exists an algorithm $\mathcal{D}$ that satisfies all of the following:*

- *It is deterministic and **non**-adaptive.*

- *Its distinguishing power between D_1 and D_2 is at least $p_{\mathcal{C}}$.*

- *Just as $\mathcal{C}$ it makes q queries. We call them $z^1, ..., z^q$.*

- *For each of these z^j, it is the case that $L(z^j) = n/2 + k\sqrt{n \log n}$.*

Proof: Consider the queries that $\mathcal{C}$ makes along the path A. Call them $y^1, ..., y^q$. We define z^j as an arbitrary point that satisfies (i) $L(z^j) = n/2 + k\sqrt{n \log n}$ and (ii) $y^j \preceq z^j$. At least one such point has to exist since $L(y^j) \leq n/2 + k\sqrt{n \log n}$.

The algorithm $\mathcal{D}$ queries each of these z^j and returns 2 if for at least one of them $f(z^j) = 1$. Otherwise it returns 1.

Since $y^j \preceq z^j$ and the functions are monotone, whenever $f(y^j) = 1$, then $f(z^j) = 1$. At least one of $f(y^j)$ equals one with probability p_{get1}, and therefore at least one of $f(z^j)$ equals one with probability at least p_{get1}.

Thus, $\mathcal{D}$ has a distinguishing power of at least $\frac{1}{2}(1 + p_{\text{get1}})$. This implies that the distinguishing power of $\mathcal{D}$ is at least that of $\mathcal{C}$. ■

Now, we can bound the distinguishing power of $\mathcal{D}$, which we call $p_{\mathcal{D}}$. We have:

$$\begin{aligned}\frac{1}{2}\Pr[\mathcal{D} \text{ returns } 2 | f \in_R D_2] = \Pr_{f \in_R D_2}\left[\bigvee_{j=0}^{q} f(z^j) = 1\right] &\leq \sum_{j=0}^{q} \Pr_{f \in_R D_2}\left[f(z^j) = 1\right] \\ &= \sum_{j=0}^{q} \Pr_{\sigma \text{ is a random permutation}}\left[F(\sigma(z^j)) = 1\right] = \\ &q \cdot \Pr_{x \in_R \{0,1\}^n}\left[F(x) = 1 \middle| L(x) = n/2 + k\sqrt{n \log n}\right] \leq \frac{q}{q_0}\end{aligned}$$

Above we used (i) a union bound and the fact that by Lemma 4.1.3, the algorithm $\mathcal{D}$ is non-adaptive (ii) The fact that by definition $f(x) = F(\sigma(x))$. Recall that σ is the random permutation F was permuted with. (iii) For any constant x, $\sigma(x)$ is uniformuly distributed among the vertices with the same level. (iv) Lemma 4.1.3 together with the condition on the function F.

Therefore, we get that:

$$p_{\mathcal{D}} \stackrel{\text{def}}{=} \frac{1}{2}\Pr[\mathcal{D} \text{ returns } 2 | f \in_R D_2] + \frac{1}{2}\Pr[\mathcal{D} \text{ returns } 1 | f \in_R D_1] \leq \frac{q}{q_0} + \frac{1}{2}$$

Since $q = o(q_0)$, then $p_{\mathcal{D}}$ has to be less than $2/3$. Since $p_{\mathcal{C}}$ is at most $p_{\mathcal{D}}$ by Lemma 4.1.3, then $p_{\mathcal{C}}$ also has to be less than $2/3$. This proves the lemma.

4.2 Proof of Lemma 4.0.4

Consider the distribution[1] H of functions, which is OR of $1/n^{C_2} \cdot 2^{\sqrt{n}}$ AND clauses of uniformly and independently chosen subsets of $\sqrt{n}$ indices, chosen without replacement. We will prove that:

a) **Any** f in H has the property that:

$$\Pr_{x \in_R \{0,1\}^n}[f(x) = 1 | L(x) = n/2 + k\sqrt{n \log n}] \leq \Theta\left(1/n^{C_2} \cdot e^{\sqrt{C_1 \log n/2}}\right)$$

b) $E_{f \in_R H}[B[f]] \geq \Omega(1/n^{C_2})$.

c) $E_{f \in_R H}[I[f]] \geq \Omega(\sqrt{n}/n^{C_2})$.

d) For any $1/n \leq \delta \leq 1/2$, it is the case that:

$$E_{f \in_R H}[NS_\delta[f]] \geq \begin{cases} \Omega(\delta\sqrt{n}/n^{C_2}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Omega(1/n^{C_2}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases}$$

Then, the corresponding claims of the lemma will follow by an application of the probabilistic method. We have divided into subsections the proofs of the claims above.

4.3 Proof of (a)

Here we treat the clauses as fixed and look at the probability over the randomness of choosing x to satisfy any of them. For a given clause, the probability that x will satisfy it equals:

[1] There are two differences between this distribution and Talagrand random functions: (i) Here we choose $1/n^{C_2} \cdot 2^{\sqrt{n}}$ clauses, whereas Talagrand functions have just $2^{\sqrt{n}}$ clauses. (ii) In Talagrand functions the indices in each clause are sampled with replacement, whereas here we sample them without replacement.

$$\prod_{i=0}^{\sqrt{n}-1} \frac{n/2 + k\sqrt{n \log n} - i}{n} \leq \left(\frac{n/2 + k\sqrt{n \log n}}{n} \right)^{\sqrt{n}}$$
$$= \frac{1}{2^{\sqrt{n}}} \left(1 + \frac{2k\sqrt{\log n}}{\sqrt{n}} \right)^{\sqrt{n}} \leq \frac{1}{2^{\sqrt{n}}} \Theta \left(e^{2k\sqrt{\log n}} \right) \leq \frac{1}{2^{\sqrt{n}}} \cdot \Theta(e^{\sqrt{C_1 \log n/2}})$$

In the very end we used that by Lemma 4.0.2 it is the case that $k \leq \sqrt{C_1/8}$.

Now, that we know the probability for one clause, we can upper-bound the probability x satisfies any of the $\frac{1}{n^{C_2}} \cdot 2^{\sqrt{n}}$ using a union bound. This gives us an upper bound of $\Theta \left(1/n^{C_2} \cdot e^{\sqrt{C_1 \log n/2}} \right)$.

4.4 Proof of b)

It is the case that:

$$E_{f \in_R H}[B[f]] = E_{f \in_R H}[E_{x \in_R \{0,1\}^n}[f(x)]] = E_{x \in_R \{0,1\}^n}[E_{f \in_R H}[f(x)]] \geq$$
$$\frac{1}{2} E_{x \in_R \{0,1\}^n} \left[E_{f \in_R H}[f(x)] \middle| L(x) \geq \frac{n}{2} \right]$$

If we fix a value of x for which $L(x) \geq n/2$, and randomly choose a single AND of $\sqrt{n}$ indices, the probability that it evaluates to one on x is:

$$\frac{L(x)}{n} \cdot \frac{L(x)-1}{n-1} \cdot \ldots \cdot \frac{L(x)-\sqrt{n}+1}{n-\sqrt{n}+1} \geq \left(\frac{L(x)-\sqrt{n}+1}{n-\sqrt{n}+1} \right)^{\sqrt{n}} \geq \left(\frac{n/2-\sqrt{n}+1}{n-\sqrt{n}+1} \right)^{\sqrt{n}}$$
$$\geq \left(\frac{1}{2} - \frac{\sqrt{n}/2 - 1/2}{n-\sqrt{n}+1} \right)^{\sqrt{n}} \geq \frac{1}{2^{\sqrt{n}}} \left(1 - \frac{1}{\Theta(\sqrt{n})} \right)^{\sqrt{n}} = \frac{1}{\Theta(2^{\sqrt{n}})}$$

Then, since we have $2^{\sqrt{n}}/n^{C_2}$ clauses and they are chosen independently:

$$E_{x\in_R\{0,1\}^n}\left[E_{f\in_R H}[f(x)]\middle| L(x)\geq\frac{n}{2}\right]\geq 1-\left(1-\frac{1}{\Theta(2^{\sqrt{n}})}\right)^{2^{\sqrt{n}}/n^{C_2}}\geq\frac{1}{\Theta(n^{C_2})}$$

This implies that $E_{f\in_R H}[B[f]]\geq 1/\Theta(n^{C_2})$.

4.5 Proof of c)

We have that:

$$E_{f\in_R H}[I[f]]=E_{f\in_R H}\left[n\cdot\Pr_{x\in_R\{0,1\}^n,i\in_R[n]}[f(x)\neq f(x^{\oplus i})]\right]=$$
$$n\cdot\Pr_{f\in_R H,x\in_R\{0,1\}^n,i\in_R[n]}[f(x)\neq f(x^{\oplus i})] \quad (4.1)$$

The probability of an event is the expectation of its indicator random variable. Using this twice, gives us the second equality above.

From Hoeffding's inequality, it follows that with probability at least 0.95 it is the case that $n/2-\sqrt{n}\leq L(x)\leq n/2+\sqrt{n}$. From this and Equation (4.1) it follows:

$$\begin{aligned}E_{f\in_R H}[I[f]]&\geq n\cdot\Pr_{f\in_R H,x\in_R\{0,1\}^n,i\in_R[n]}\Bigg[f(x)\neq f(x^{\oplus i})\Bigg|\\&\quad n/2-\sqrt{n}\leq L(x)\leq n/2+\sqrt{n}\Bigg]\cdot\Pr_{x\in_R\{0,1\}^n}[n/2-\sqrt{n}\leq L(x)\leq n/2+\sqrt{n}]\\&\geq 0.95n\Pr_{f\in_R H,x\in_R\{0,1\}^n,i\in_R[n]}\left[f(x)\neq f(x^{\oplus i})\middle| n/2-\sqrt{n}\leq L(x)\leq n/2+\sqrt{n}\right] \quad (4.2)\end{aligned}$$

Now we will lower-bound $\Pr_{f\in_R H,i\in_R[n]}[f(x)\neq f(x^{\oplus i})]$ for any x, for which $n/2-\sqrt{n}\leq L(x)\leq n/2+\sqrt{n}$. Name the clauses in f as $\wedge_1,\wedge_2,...,\wedge_{\frac{1}{n^{C_2}}2^{\sqrt{n}}}$. For any clause $\wedge_j$ we have:

$$\Pr_{f\in_R H}[\wedge_j\text{ is satisfied}]=\frac{L(x)}{n}\cdot\frac{L(x)-1}{n-1}\cdot...\cdot\frac{L(x)-\sqrt{n}+1}{n-\sqrt{n}+1}$$

And since $n/2 - \sqrt{n} \leq L(x) \leq n/2 + \sqrt{n}$:

$$\frac{1}{2^{\sqrt{n}}}\left(1 - \frac{1}{\Theta(\sqrt{n})}\right)^{\sqrt{n}} \leq \left(\frac{L(x) - \sqrt{n} + 1}{n - \sqrt{n} + 1}\right)^{\sqrt{n}} \leq \Pr_{f \in_R H}[\wedge_j \text{ is satisfied}] \leq \left(\frac{L(x)}{n}\right)^{\sqrt{n}} \leq \frac{1}{2^{\sqrt{n}}}\left(1 + \frac{1}{\Theta(\sqrt{n})}\right)^{\sqrt{n}}$$

This implies that:

$$\Pr_{f \in_R H}[\wedge_j \text{ is satisfied}] = \frac{1}{\Theta(2^{\sqrt{n}})} \tag{4.3}$$

For every i, so that $1 \leq i \leq 2^{\sqrt{n}}/n^{C_2}$, consider the following sequence of events, which we call M_i:

- x satisfies $\wedge_i$. By Equation (4.3) the probability of this happening is $1/\Theta(2^{\sqrt{n}})$.
- x does not satisfy all the other $\frac{1}{n^{C_2}}2^{\sqrt{n}} - 1$ clauses.
  Since the clauses are chosen independently, by Equation (4.3) we have that the probability of this is $\left(1 - 1/\Theta(2^{\sqrt{n}})\right)^{2^{\sqrt{n}}/n^{C_2}-1}$, which is at least $\Theta(1)$.
- i is one of the inputs that are relevant to $\wedge_i$. The probability of this is $1/\sqrt{n}$.

Since these three events are independent:

$$\Pr_{f \in_R H}[M_i] \geq \frac{1}{\Theta(2^{\sqrt{n}})} \cdot \Theta(1) \cdot \frac{1}{\sqrt{n}} = \frac{1}{\Theta(\sqrt{n}2^{\sqrt{n}})}$$

If M_i happens, then $f(x) \neq f(x^{\oplus i})$. Additionally for different values of i, the M_i are disjoint and the probability of M_i is the same for all i by symmetry. Thus we have:

$$\Pr_{f \in_R H, i \in_R [n]}\left[f(x) \neq f(x^{\oplus i})\right] \geq \Pr_{f \in_R H, i \in_R [n]}\left[\bigvee_i M_i\right] = \frac{2^{\sqrt{n}}}{n^{C_2}}\Pr_{f \in_R H, i \in_R [n]}[M_1] \geq \frac{2^{\sqrt{n}}}{n^{C_2}} \cdot \frac{1}{\Theta(\sqrt{n}2^{\sqrt{n}})} = \frac{1}{\Theta(\sqrt{n} \cdot n^{C_2})} \tag{4.4}$$

Combining Equations (4.2) and (4.4):

$$E_{f \in_R H}[I[f]] \geq 0.95n \cdot \frac{1}{\Theta(\sqrt{n} \cdot n^{C_2})} = \Omega\left(\frac{\sqrt{n}}{n^{C_2}}\right)$$

4.6 Proof of d)

Recall that in the definition of noise sensitivity, x is chosen uniformly and y is chosen by flipping each bit of x with probability δ. We have:

$$E_{f\in_R H}[NS_\delta[f]] = E_{f\in_R H}\left[\Pr_{(x,y)\in_R T_\delta}[f(x)\neq f(y)]\right] = \Pr_{f\in_R H,(x,y)\in_R T_\delta}[f(x)\neq f(y)] \tag{4.5}$$

Consider the following three "good" events, which are similar to the ones introduced in [18] to analyze the noise sensitivity of Talagrand random functions:

- G_1 is when $n/2-\sqrt{n}\leq L(x)\leq n/2+\sqrt{n}$. By Hoeffding's inequality, its probability is at least 0.95.
- G_2 is when $n/2-\sqrt{n}\leq L(y)\leq n/2+\sqrt{n}$. Since y is also distributed uniformly, its probability is also at least 0.95.
- Denote by S_x the set of indices i for which $x_i = 1$. By the definition of noise sensitivity, in expectation $\delta|S_x|$ of them become zero in y. The event G_3 happens when at least $\delta|S_x|/2$ of them are zero in y. By the Chernoff bound, the probability of this is at least $1-\exp(-\delta n/8)\geq 1-\exp(-1/8)\geq 0.11$.

By a union bound, with probability at least 0.01 all the events G_1, G_2 and G_3 happen. Therefore:

$$\begin{aligned} &E_{f\in_R H}[NS_\delta[f]] \geq \Pr_{f\in_R H,(x,y)\in_R T_\delta}\left[f(x)\neq f(y)\middle| G_1\wedge G_2\wedge G_3\right] \\ &\times \Pr_{(x,y)\in_R T_\delta}[G_1\wedge G_2\wedge G_3] \geq 0.01\cdot \Pr_{f\in_R H,(x,y)\in_R T_\delta}\left[f(x)\neq f(y)\middle| G_1\wedge G_2\wedge G_3\right] \end{aligned} \tag{4.6}$$

Now, suppose we are given values of x and y that satisfy $G_1\wedge G_2\wedge G_3$, we will lower bound $\Pr_{f\in_R H}[f(x)\neq f(y)]$. Using G_1 and G_2, just as in the proof of equation (5), we

have that for any clause $\wedge_j$:

$$\Pr_{f\in_R H}[\wedge_j \text{ is satisfied by } x] = \frac{1}{\Theta(2^{\sqrt{n}})} \quad \Pr_{f\in_R H}[\wedge_j \text{ is satisfied by } y] = \frac{1}{\Theta(2^{\sqrt{n}})} \tag{4.7}$$

Now, analogous to how we lower bounded the expected influence while proving part c), consider the following sequence of events, which we call N_i:

- x satisfies $\wedge_i$. By Equation (4.7), the probability of this is at least $1/\Theta(2^{\sqrt{n}})$.
- All the $\wedge_j$ for $j \neq i$ are unsatisfied by both x and y. By Equation (4.7) and a union bound, for each individual clause the probability of being unsatisfied by both x and y is at least $1 - 2 \cdot 1/\Theta(2^{\sqrt{n}}) = 1 - 1/\Theta(2^{\sqrt{n}})$. By independence, the probability of the overall event is at least $\left(1 - 1/\Theta(2^{\sqrt{n}})\right)^{2^{\sqrt{n}}/n^{C_2}-1}$, which is at least $\Theta(1)$.
- Given that x satisfies $\wedge_i$, it happens that at least one of the coordinates relevant to $\wedge_i$ is zero in y. We call the probability of this happening p_{flip}.

The third event is conditioned on the first one, so the probability that both happen equals to the product of their probabilities. In addition, The first and third event depend only on the randomness in choosing $\wedge_i$, and the second event only on the randomness in choosing all the other clauses. Therefore the second event is independent from the first and third, and thus:

$$\Pr_{f\in_R H}[N_i] \geq \frac{1}{\Theta(2^{\sqrt{n}})} \cdot \Theta(1) \cdot p_{\text{flip}} = \frac{p_{\text{flip}}}{\Theta(2^{\sqrt{n}})} \tag{4.8}$$

We now lower-bound p_{flip}. Because of G_3, at least $\delta|S_x|/2$ of the indices in S_x become zero in y. Let $S_{\wedge_i}$ be the set of $\sqrt{n}$ indices relevant to $\wedge_i$. Since they were chosen uniformly at random then, conditioning on $\wedge_i$ being satisfied, $S_{\wedge_i}$ has equal probability of being any subset of S_x of size $\sqrt{n}$. Therefore, the probability that at least one of them ends up among the indices in S_x that become zero in y:

$$p_{\text{flip}} = 1 - \prod_{j=0}^{\sqrt{n}-1}\left(1 - \frac{\delta|S_x|/2}{|S_x| - j}\right) \geq 1 - (1-\delta/2)^{\sqrt{n}} \geq \begin{cases} \Theta(\delta\sqrt{n}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Theta(1) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases}$$

Combining this with Equation (4.8), we get:

$$\Pr_{f\in_R H}[N_i] \geq \frac{p_{\text{flip}}}{\Theta(2^{\sqrt{n}})} \geq \begin{cases} \Theta(\delta\sqrt{n}/2^{\sqrt{n}}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Theta(1/2^{\sqrt{n}}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases} \tag{4.9}$$

We now combine (i) Equation (4.6) (ii) The fact that if N_i happens then $f(x) \neq f(y)$ (iii) The fact that the different N_i are disjoint and the probability of N_i is the same for all i by symmetry (iv) Equation (4.9):

$$E_{f\in_R H}[NS_\delta[f]] \geq 0.01 \cdot \Pr_{f\in_R H}\left[\bigvee_i N_i\right] = 0.01 \cdot \frac{2^{\sqrt{n}}}{n^{C_2}} \Pr_{f\in_R H, i\in_R[n]}[N_1] \geq$$
$$\begin{cases} \Theta(\delta\sqrt{n}/n^{C_2}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Theta(1/n^{C_2}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases}$$

Bibliography

[1] Maria-Florina Balcan, Eric Blais, Avrim Blum, and Liu Yang. Active property testing. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 21–30. IEEE, 2012.

[2] Aleksandrs Belovs and Eric Blais. A polynomial lower bound for testing monotonicity. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1021–1032. ACM, 2016.

[3] Itai Benjamini, Gil Kalai, and Oded Schramm. Noise sensitivity of boolean functions and applications to percolation. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 90(1):5–43, 1999.

[4] Eric Blais, Ryan O'Donnell, and Karl Wimmer. Polynomial regression under arbitrary product distributions. *Machine learning*, 80(2-3):273–294, 2010.

[5] Deeparnab Chakrabarty and Comandur Seshadhri. An o(n) monotonicity tester for boolean functions over the hypercube. *SIAM Journal on Computing*, 45(2):461–472, 2016.

[6] Xi Chen, Rocco A Servedio, and Li-Yang Tan. New algorithms and lower bounds for monotonicity testing. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 286–295. IEEE, 2014.

[7] Mahdi Cheraghchi, Adam Klivans, Pravesh Kothari, and Homin K Lee. Submodular functions are noise stable. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1586–1592. Society for Industrial and Applied Mathematics, 2012.

[8] Ilias Diakonikolas, Prasad Raghavendra, Rocco A. Servedio, and Li-Yang Tan. Average sensitivity and noise sensitivity of polynomial threshold functions. *SIAM J. Comput.*, 43(1):231–253, 2014.

[9] Adam Tauman Kalai, Adam R Klivans, Yishay Mansour, and Rocco A Servedio. Agnostically learning halfspaces. *SIAM Journal on Computing*, 37(6):1777–1805, 2008.

[10] Gil Kalai et al. Noise sensitivity and chaos in social choice theory. Technical report, 2005.

[11] Daniel M. Kane. The gaussian surface area and noise sensitivity of degree-*d* polynomial threshold functions. *Computational Complexity*, 20(2):389–412, 2011.

[12] Daniel M. Kane. The average sensitivity of an intersection of half spaces. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 437–440, 2014.

[13] Nathan Keller and Guy Kindler. Quantitative relation between noise sensitivity and influences. *Combinatorica*, 33(1):45–71, 2013.

[14] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[15] Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric type theorems. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 52–58. IEEE, 2015.

[16] Adam R Klivans, Ryan O'Donnell, and Rocco A Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer and System Sciences*, 68(4):808–840, 2004.

[17] Rajsekar Manokaran, Joseph Seffi Naor, Prasad Raghavendra, and Roy Schwartz. SDP gaps and UGC hardness for multiway cut, 0-extension, and metric labeling. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 11–20. ACM, 2008.

[18] Elchanan Mossel and Ryan O'Donnell. On the noise sensitivity of monotone functions. *Random Structures & Algorithms*, 23(3):333–350, 2003.

[19] Elchanan Mossel and Ryan O'Donnell. Coin flipping from a cosmic source: On error correction of truly random bits. *Random Structures & Algorithms*, 26(4):418–436, 2005.

[20] Ryan O'Donnell. Hardness amplification within NP. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 751–760. ACM, 2002.

[21] Ryan O'Donnell. *Computational applications of noise sensitivity*. PhD thesis, Massachusetts Institute of Technology, 2003.

[22] Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

[23] Yuval Peres. Noise stability of weighted majority. *arXiv preprint math/0412377*, 2004.

[24] Dana Ron, Ronitt Rubinfeld, Muli Safra, Alex Samorodnitsky, and Omri Weinstein. Approximating the influence of monotone boolean functions in $O(\sqrt{n})$ query complexity. *TOCT*, 4(4):11:1–11:12, 2012.

[25] Dana Ron, Ronitt Rubinfeld, Muli Safra, and Omri Weinstein. Approximating the influence of monotone boolean functions in $O(\sqrt{n})$ query complexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 664–675. Springer, 2011.

[26] Michel Talagrand. How much are increasing sets positively correlated? *Combinatorica*, 16(2):243–258, 1996.

Appendix A

Appendix A

In this section we use Fourier analysis of boolean functions. We will use the notation of [22].

The following lemma is very similar to a statement from [18]:

Lemma A.0.1 *For any function* $f\colon \{0,1\}^n \to \{0,1\}$ *and a parameter* $\delta \leq 1/2$ *it is the case that:*

$$NS_\delta[f] \leq \delta I[f]$$

Proof: We will use the Fourier expressions for both of the above (see [22]):

$$I[f] = \sum_S |S|\hat{f}^2(S) \qquad\qquad NS_\delta[f] = \frac{1}{2}\sum_S (1-(1-2\delta)^{|S|})\hat{f}^2(S)$$

We can now use Bernoulli's inequality $(1-2\delta)^{|S|} \geq 1-2\delta|S|$. Therefore:

$$NS_\delta[f] \leq \frac{1}{2}\sum_S 2\delta|S|\hat{f}^2(S) = \delta\sum_S |S|\hat{f}^2(S) = \delta I[f]$$

This completes the proof. ■

Lemma A.0.2 *For a fixed function* $f\colon \{0,1\}^n \to \{0,1\}$ *and for values of* δ *satisfying* $0 < \delta \leq 1/2$, $NS_\delta[f]$ *is an increasing function of* δ.

Proof: This follows immediately from the Fourier formula for noise sensitivity. ■

Appendix B

A query complexity lower bound for general Boolean functions

Recall that standard sampling approach requires $O(\frac{1}{NS_\delta[f]\epsilon^2})$ queries to estimate noise sensitivity. Here we will show that for sufficiently small constant ϵ, the standard sampling algorithm is optimal up to a constant for all values of $NS_\delta[f] \geq 1/2^n$.

For any α we define H^α to be the uniform distribution over all functions $f : \{0,1\}^n \to \{0,1\}$ for which it is the case that $\Pr_{x \in_R \{0,1\}^n}[f(x) = 1] = \alpha$.

Lemma B.0.1 *For any sufficiently large n, any α, satisfying $10^6/2^n \leq \alpha \leq 1/2$ and any δ, satisfying $1/n \leq \delta \leq 1/2$, it is the case that:*

$$\Pr_{f \in_R H^\alpha}[0.1\alpha \leq NS_\delta[f] \leq 3\alpha] \geq 0.99$$

Proof: $x = y$ implies that $f(x) = f(y)$, hence:

$$\begin{aligned} E_{f \in_R H^\alpha}[NS_\delta[f]] &= \Pr_{f \in_R H^\alpha; (x,y) \in_R T_\delta}[f(x) \neq f(y)] \\ &= \Pr_{(x,y) \in_R T_\delta}[x \neq y] \cdot \Pr_{f \in_R H^\alpha; (x,y) \in_R T_\delta}[f(x) \neq f(y) | x \neq y] \quad \text{(B.1)} \end{aligned}$$

Since $\delta \geq 1/n$, we have that:

$$\Pr_{(x,y)\in_R T_\delta}[x \neq y] = 1 - (1-\delta)^n \geq 1 - (1-1/n)^n \geq 1 - e^{-1} \geq 0.25 \tag{B.2}$$

Additionally, we have:

$$\Pr_{f\in_R H^\alpha;(x,y)\in_R T_\delta}[f(x) \neq f(y) | x \neq y] = 2\alpha \cdot \frac{2^n \cdot (1-\alpha)}{2^n - 1} \tag{B.3}$$

Combing equations (B.1), (B) and (B.3), we get:

$$E_{f\in_R H^\alpha}[NS_\delta[f]] = \Pr_{(x,y)\in_R T_\delta}[x \neq y] \cdot 2\alpha \cdot \frac{2^n \cdot (1-\alpha)}{2^n - 1} \geq 0.20\alpha \tag{B.4}$$

At the same time, for sufficiently large n we have that:

$$E_{f\in_R H^\alpha}[NS_\delta[f]] = \Pr_{(x,y)\in_R T_\delta}[x \neq y] \cdot 2\alpha \cdot \frac{2^n \cdot (1-\alpha)}{2^n - 1} \leq 2.5\alpha \tag{B.5}$$

Now, we will bound the variance. We have:

$$\begin{aligned} E_{f\in_R H^\alpha}[(NS_\delta[f])^2] &= \Pr_{f\in_R H^\alpha;(x^1,y^1)\in_R T_\delta;(x^2,y^2)\in_R T_\delta}[(f(x^1) \neq f(y^1)) \wedge (f(x^2) \neq f(y^2))] \\ = (\Pr_{(x,y)\in_R T_\delta}[x \neq y])^2 \cdot &\Pr_{f\in_R H^\alpha;(x^1,y^1)\in_R T_\delta;(x^2,y^2)\in_R T_\delta}\Big[(f(x^1) \neq f(y^1)) \wedge (f(x^2) \neq f(y^2)) \Big| \\ &\qquad (x^1 \neq y^1) \wedge (x^2 \neq y^2)\Big] \end{aligned} \tag{B.6}$$

We have the following three facts:

1. For arbitrary x^1, y^1, x^2 and y^2, we have

$$\begin{aligned} &\Pr_{f\in_R H^\alpha}[(f(x^1) \neq f(y^1)) \wedge (f(x^2) \neq f(y^2))] \\ &\qquad \leq \Pr_{f\in_R H^\alpha}[(f(x^1) \neq f(y^1)] \leq 2\alpha \cdot 2^n(1-\alpha)/(2^n - 1) \end{aligned}$$

2. Suppose we further given that no two of x^1, x^2, y^1 and y^2 are equal to each other. We

call this event K. If K is the case then

$$\Pr_{f\in_R H^\alpha}[(f(x^1)\neq f(y^1))\wedge(f(x^2)\neq f(y^2))] = 4\alpha\cdot\frac{2^n(1-\alpha)}{2^n-1}\cdot\frac{2^n\alpha-1}{2^n-2}\cdot\frac{2^n(1-\alpha)-1}{2^n-3}$$

3. If (x^1,y^1) and (x^2,y^2) are picked independently from T_δ conditioned on $x^1\neq y^1$ and $x^2\neq y^2$, then K happens unless $x^1=x^2$ or $x^1=y^2$ or $y^1=x^2$ or $y^1=y^2$. By independence and a union bound, the probability of any of these happening is at most $4/2^n$.

Therefore:

$$\begin{aligned}
&\Pr_{f\in_R H^\alpha;(x^1,y^1)\in_R T_\delta;(x^2,y^2)\in_R T_\delta}\left[(f(x^1)\neq f(y^1))\wedge(f(x^2)\neq f(y^2))\middle|(x^1\neq y^1)\wedge(x^2\neq y^2)\right]\\
&\quad\leq \Pr_{f\in_R H^\alpha;(x^1,y^1)\in_R T_\delta;(x^2,y^2)\in_R T_\delta}\Bigg[(f(x^1)\neq f(y^1))\wedge(f(x^2)\neq f(y^2))\Bigg|\\
&\quad(x^1\neq x^2)\wedge(x^1\neq y^1)\wedge(y^1\neq x^2)\wedge(y^1\neq y^2)\wedge(x^1\neq y^1)\wedge(x^2\neq y^2)\Bigg]\\
&\quad+\Bigg(\Pr_{f\in_R H^\alpha;(x^1,y^1)\in_R T_\delta;(x^2,y^2)\in_R T_\delta}\Bigg[(f(x^1)\neq f(y^1))\wedge(f(x^2)\neq f(y^2))\Bigg|\\
&\qquad(x^1=x^2)\vee(x^1=y^1)\vee(y^1=x^2)\vee(y^1=y^2)\Bigg]\\
&\quad\times\Pr_{(x^1,y^1)\in_R T_\delta;(x^2,y^2)\in_R T_\delta}[(x^1=x^2)\vee(x^1=y^1)\vee(y^1=x^2)\vee(y^1=y^2)]\Bigg)\\
&\quad\leq 4\alpha\cdot\frac{2^n(1-\alpha)}{2^n-1}\cdot\frac{2^n\alpha-1}{2^n-2}\cdot\frac{2^n(1-\alpha)-1}{2^n-3}+2\alpha\cdot 2^n\frac{1-\alpha}{2^n-1}\frac{4}{2^n}
\end{aligned}\tag{B.7}$$

Combining this with (B.4) and (B.6) we get a bound for the variance:

$$\begin{aligned}
Var_{f\in_R H^\alpha}[NS_\delta[f]] &= (\Pr_{(x,y)\in_R T_\delta}[x\neq y])^2\cdot\Bigg(4\alpha\cdot\frac{2^n(1-\alpha)}{2^n-1}\cdot\frac{2^n\alpha-1}{2^n-2}\cdot\frac{2^n(1-\alpha)-1}{2^n-3}+\\
&\qquad 2\alpha\frac{2^n\cdot(1-\alpha)}{2^n-1}\frac{4}{2^n}-\left(2\alpha\cdot\frac{2^n\cdot(1-\alpha)}{2^n-1}\right)^2\Bigg)\\
&\leq\frac{4\alpha^2(1-\alpha)^2}{(1-1/2^n)\cdot(1-2/2^n)\cdot(1-3/2^n)}+\frac{\alpha(1-\alpha)}{(1-1/2^n)}\cdot\frac{8}{2^n}-4\alpha^2(1-\alpha)^2\leq\\
&\qquad\frac{48}{2^n}\alpha^2(1-\alpha)^2+\frac{16}{2^n}\alpha(1-\alpha)\leq\frac{100\alpha}{2^n}
\end{aligned}\tag{B.8}$$

(B.8) implies that $NS_\delta[f]$ has standard deviation of at most $10\sqrt{\alpha/2^n}$. Since $\alpha \geq 10^6/2^n$, this standard deviation of $NS_\delta[f]$ is at most $\alpha/100$. By Chebyshev's inequality, $NS_\delta[f]$ is within $\alpha/10$ of its expectation with probability 0.99. Together with equations (B.4) and (B.5) this implies the statement of the lemma. ■

Theorem 6 *For any sufficiently large n, any δ satisfying $1/n \leq \delta \leq 1/2$ and any α_0 satisfying $10^5/2^n \leq \alpha_0 \leq 1/1200$, let $\mathcal{G}$ be an algorithm that given access to a function $f\colon \{0,1\}^n \to \{0,1\}$ with probability at least 0.99 outputs NO if it is given access to a function $f\colon \{0,1\}^n \to \{0,1\}$, satisfying*

$$\alpha_0 \leq NS_\delta[f] \leq 30\alpha_0$$

and with probability at least 0.99 outputs YES, given a function satisfying:

$$60\alpha_0 \leq NS_\delta[f] \leq 1800\alpha_0$$

Then, there is a function f_0 given which $\mathcal{G}$ makes $\Omega\left(\frac{1}{\alpha_0}\right) = \Omega\left(\frac{1}{NS_\delta[f]}\right)$ queries.

Proof: Consider distributions $H^{10\alpha_0}$ and $H^{600\alpha_0}$. One needs $\Omega\left(\frac{1}{\alpha_0}\right) = \Omega\left(\frac{1}{NS_\delta[f]}\right)$ queries to distinguish between them with any constant probability. Both values $10\alpha_0$ and $600\alpha_0$ are within the scope of Lemma B.0.1. Therefore, from Lemma B.0.1 it is the case that:

$$\Pr_{f\in_R H^{\alpha_0/B_1}}\left[\alpha_0 \leq NS_\delta[f] \leq 30\alpha_0\right] \geq 0.99$$

$$\Pr_{f\in_R H^{2B_2\alpha_0/B_1^2}}\left[60\alpha_0 \leq NS_\delta[f] \leq 1800\alpha_0\right] \geq 0.99$$

Since $\mathcal{G}$ is correct with probability at least 0.99, by a union bound it will distinguish between a random function from H^{α_0/B_1} and $H^{2B_2\alpha_0/B_1^2}$ with probability at least 0.98. But one needs at least $\Omega\left(\frac{1}{\alpha_0}\right) = \Omega\left(\frac{1}{NS_\delta[f]}\right)$ queries to distinguish them. This implies the lower bound on the number of queries $\mathcal{G}$ makes. ■

Appendix C

Proofs of technical lemmas pertaining to the algorithm

C.1 Appendix C

C.2 Proof of Lemma 2.1.1

We distinguish three cases:

1. $l_1 \geq \frac{n}{2}$

2. $l_1 \leq \frac{n}{2} \leq l_2$

3. $l_2 \leq \frac{n}{2}$.

We first prove the case 1. Since here $\binom{n}{l_1} \geq \binom{n}{l_2}$, the left inequality is true. We proceed to prove the right inequality. So, for sufficiently large n:

$$\frac{\binom{n}{l_1}}{\binom{n}{l_2}} = \frac{l_2!(n-l_2)!}{l_1!(n-l_1)!} = \prod_{i=0}^{l_2-l_1-1} \frac{l_2-i}{n-l_1-i} \leq \left(\frac{l_2}{n-l_2}\right)^{l_2-l_1} \leq \left(\frac{\frac{n}{2}+\sqrt{C_1 n\log(n)}}{\frac{n}{2}-\sqrt{C_1 n\log(n)}}\right)^{l_2-l_1}$$

$$\leq \left(1+5\sqrt{\frac{C_1\log(n)}{n}}\right)^{l_2-l_1} \leq \left(1+5\sqrt{\frac{C_1\log(n)}{n}}\right)^{C_2\xi\sqrt{\frac{n}{\log(n)}}} \leq e^{5C_2\sqrt{C_1}\xi} = e^{0.5\xi} \leq 1+\xi$$

This completes the proof for case 1. We note that this method of bounding the product above was inspired by the proof in [25]. There it was used to bound probabilities of random walks directly, whereas we are using it to proof this Continuity Lemma first and then apply it later for random walks.

Now, we derive case 3 from it. Suppose $l_1 \leq l_2 \leq \frac{n}{2}$, then, $\binom{n}{l_1} \leq \binom{n}{l_2}$ which gives us the inequality on the right. To prove the left one, define $l_2' = n - l_1$ and $l_1' = n - l_2$. l_1' and l_2' will satisfy all the requirements for case 1, thus we have:

$$\frac{\binom{n}{l_1'}}{\binom{n}{l_2'}} \leq 1+\xi$$

This implies:

$$\frac{\binom{n}{l_1}}{\binom{n}{l_2}} = \frac{\binom{n}{l_2'}}{\binom{n}{l_1'}} \geq \frac{1}{1+\xi} \geq 1-\xi$$

This completes the proof for case 3. For case 2, together cases 1 and 3 imply that the following are true:

$$1 \leq \frac{\binom{n}{n/2}}{\binom{n}{l_2}} \leq 1+\xi \qquad\qquad 1-\xi \leq \frac{\binom{n}{l_1}}{\binom{n}{n/2}} \leq 1$$

Multiplying these together, we show the lemma in case 2.

C.3 Proof of Lemma 3.1.3

Recall that we have $NS_\delta[f] = 2 \cdot \Pr_D[f(x) = 1 \wedge f(z) = 0]$ and $p_A p_B = \Pr_D\Big[f(x) = 1 \wedge f(z) = 0 \wedge \overline{E_1} \wedge \overline{E_2}\Big]$, which implies the left inequality. We now prove the right one. We have:

$$\frac{NS_\delta[f]}{2} = \Pr_D[f(x) = 1 \wedge f(z) = 0] \leq \Pr_D[f(x) = 1 \wedge f(z) = 0 \wedge \overline{E_1} \wedge \overline{E_2}] + \Pr_D[E_1 \vee E_2] \tag{C.1}$$

By Chernoff bound we have:

$$\Pr_D[E_1] \leq \exp(-\frac{1}{3} n \cdot \delta 3\eta_2 \log n) \leq \frac{1}{n^{\eta_2}} \tag{C.2}$$

Now using the Hoeffding bound together with the fact that since $\delta \leq 1/(\sqrt{n} \log n)$ we have $t_2 \leq \sqrt{n}/\log n \cdot (1 + 3\eta_2 \log(n)) \leq \sqrt{n \log n}$:

$$\Pr_D[E_2 | \overline{E_1}] \leq \Pr_D[|L(x) - n/2| \geq t_1 - t_2] \leq 2\exp(-2(\eta_1 - 1)^2 \log n) = \frac{2}{n^{2(\eta_1-1)^2}} \tag{C.3}$$

Thus, combining Equations (C.1) and (C.2) with Observation 2.1.3 we have:

$$\Pr_D[E_1 \vee E_2] \leq \Pr_D[E_1] + \Pr_D[E_2 | \overline{E_1}] \leq \frac{1}{n^{\eta_2}} + \frac{1}{n^{2(\eta_1-1)^2}} \leq \frac{\epsilon}{15n^C} \tag{C.4}$$

Combining Equations (C.1) and (C.4) we get:

$$\frac{1}{2} NS_\delta[f] \leq \Pr_D[f(x) = 1 \wedge f(z) = 0 \wedge \overline{E_1} \wedge \overline{E_2}] + \frac{\epsilon}{15n^C}$$

Since $NS_\delta[f] \geq \frac{1}{n^C}$ and $p_A p_B = \Pr_D[f(x) = 1 \wedge f(z) = 0 \wedge \overline{E_1} \wedge \overline{E_2}]$, this implies the lemma.

C.4 Proof of Lemma 3.5.3

Recall that the probability of any given iteration to be successful is ϕ. We can upper-bound the probability that the inequality fails to hold by the sum of probabilities the two following bad events: (i) after $(1-\epsilon/16)\cdot 768\ln 200/(\epsilon^2\phi)$ iterations there are more than $768\ln 200/\epsilon^2$ successes. (ii) after $(1+\epsilon/16)\cdot 768\ln 200/(\epsilon^2\phi)$ iterations there are less than $768\ln 200/\epsilon^2$ successes.

By Chernoff bound:

$$\Pr[\text{(i) happens}] \leq \exp\left(-\frac{1}{3}\left(\frac{1}{1-\epsilon/16}-1\right)^2(1-\epsilon/16)\frac{768\ln 200}{\epsilon^2}\right) \leq 0.005$$

$$\Pr[\text{(ii) happens}] \leq \exp\left(-\frac{1}{2}\left(1-\frac{1}{1+\epsilon/16}\right)^2(1+\epsilon/16)\frac{768\ln 200}{\epsilon^2}\right) \leq 0.005$$

This proves the correctness.

To bound the expected number of iterations, we first observe that from a similar Chernoff bound, after $O(1/(\epsilon^2\phi))$ iterations with probability at lest $1/2$ we exit the main loop. Each further time we make the same number of iterations, the probability of exiting only increases. This implies that the expected number of iterations is $O(1/(\epsilon^2\phi))$.

C.5 Proof of Lemma 3.5.4

Recall that by Equation (3.3):

$$p_A p_B = \sum_{e\in E_I\cap M} p_e q_e$$

By Lemma 3.3.2:

$$\left(1-\frac{\epsilon}{310}\right)\frac{\delta}{2^n}\sum_{e\in E_I\cap M} q_e \leq p_A p_B \leq \left(1+\frac{\epsilon}{310}\right)\frac{\delta}{2^n}\sum_{e\in E_I\cap M} q_e$$

Dividing this equation by the equation in Lemma 3.5.1 and substituting $|E_I| = 2^{n-1}I[f]$:

$$\left(1-\frac{\epsilon}{70}\right)\frac{1}{|E_I|}\sum_{e\in E_I\cap M} q_e \le p_B \le \left(1+\frac{\epsilon}{70}\right)\frac{1}{|E_I|}\sum_{e\in E_I\cap M} q_e$$

Now applying Observation 3.1.2 :

$$\left(1-\frac{\epsilon}{33}\right)\frac{1}{|E_I\cap M|}\sum_{e\in E_I\cap M} q_e \le p_B \le \left(1+\frac{\epsilon}{33}\right)\frac{1}{|E_I\cap M|}\sum_{e\in E_I\cap M} q_e \tag{C.5}$$

Define Ψ to be the set of pairs of paths (P_1', P_2') for which the following hold:

- P_1' is a descending path and P_2' is an ascending path.
- The endpoint of P_1' is the starting point of P_2'.
- The value of f at the starting point of P_1' is one, and it is zero at the endpoint of P_2'.

If and only if (P_1, P_2) is in Ψ, we have that $f(x) \neq f(z)$, therefore these are the only paths contributing to ϕ.

Using this definition, we have:

$$\phi = \sum_{e\in E_I\cap M}\left(\Pr_{e'\in_R\mathcal{A}}[e'=e]\sum_{(P_1',P_2')\in\Psi:e\in P_1'}\Pr_{\mathcal{B}_e}[(P_1=P_1')\wedge(P_2=P_2')]\right) \tag{C.6}$$

$$q_e = \sum_{(P_1',P_2')\in\Psi:e\in P_1'}\Pr_D[(P_1=P_1')\wedge(P_2=P_2')|((e\in P_1))\wedge\overline{E_1}\wedge\overline{E_2}] \tag{C.7}$$

Combining Equation (C.7) and Lemma 3.4.3 we get:

$$\left(1-\frac{\epsilon}{70}\right)\sum_{(P_1',P_2')\in\Psi:e\in P_1}\Pr_{\mathcal{B}(e)}[(P_1=P_1')\wedge(P_2=P_2')] \le q_e$$
$$\le \left(1+\frac{\epsilon}{70}\right)\sum_{(P_1',P_2')\in\Psi:e\in P_1}\Pr_{\mathcal{B}(e)}[(P_1=P_1')\wedge(P_2=P_2')] \tag{C.8}$$

Now, if in Lemma 3.3.1 we fix an e_2 and sum over e_1 in $E \cap M$ we get that for all e:

$$\left(1 - \frac{\epsilon}{70}\right) \Pr_{e' \in_R \mathcal{A}}[e' = e] \leq \frac{1}{|E \cap M|} \leq \left(1 + \frac{\epsilon}{70}\right) \Pr_{e' \in_R \mathcal{A}}[e' = e] \tag{C.9}$$

Combining Equation (C.6) with Equation (C.8) and Equation (C.9) we get:

$$\left(1 - \frac{\epsilon}{33}\right) \phi \leq \frac{1}{|E_I \cap M|} \sum_{e \in E_I \cap M} q_e \leq \left(1 + \frac{\epsilon}{33}\right) \phi \tag{C.10}$$

Equations (C.5) and (C.10) together imply the lemma.

Appendix D

Proofs of technical lemmas pertaining to query complexity lower bounds

D.1 Proof of Lemma 4.0.1

If we make the function equal to one on inputs it possibly equaled zero, the bias of the function cannot decrease. Additionally, F' can be constructed from F by changing less than $1/n^{C_1}$ fraction of its points. Such a transformation cannot increase the bias by more than $1/n^{C_1}$. This proves (a).

Regarding (b) and (c), we start we the following observation: suppose only one point $f(x^0)$ of an arbitrary function is changed. By union bound the probability for a randomly chosen x that either $x = x^0$ or $x^{\oplus i} = x^0$ is at most $1/2^{n-1}$. Therefore, the probability that $f(x) \neq f(x^{\oplus i})$ cannot change by more than $1/2^{n-1}$. This implies that the influence cannot change by more than $n/2^{n-1}$.

Regarding noise sensitivity, the situation is analogous. For any δ, the probability that any of x and y equals x_0, on which the value of the function is changed, is at most $1/2^{n-1}$ by a union bound. Therefore, $NS_\delta[f]$ cannot change by more than $1/2^{n-1}$.

Using the observation and a triangle inequality we get:

$$\left|I[F] - I[F']\right| \leq \frac{2^n}{n^{C_1}} \cdot \frac{n}{2^{n-1}} = \frac{2n}{n^{C_1}}$$

$$\left|NS_\delta[F] - NS_\delta[F']\right| \leq \frac{2^n}{n^{C_1}} \cdot \frac{1}{2^{n-1}} = \frac{2}{n^{C_1}}$$

This completes the proof of (b) and (c).

D.2 Proof of Lemma 4.0.2

Now, we proceed to proving (a). By our condition on k, it has to be the case that:

$$\Pr_{x \in_R \{0,1\}^n}[L(x) \geq n/2 + k\sqrt{n \log n}] \geq \frac{1}{n^{C_1}} \tag{D.1}$$

Then, by Hoeffding's bound, it is the case that:

$$\frac{1}{n^{C_1}} \leq \Pr_{x \in_R \{0,1\}^n}[L(x) \geq n/2 + k\sqrt{n \log n}] \leq \exp\left(-2n \cdot \left(\frac{2k\sqrt{n \log n}}{n}\right)^2\right)$$
$$= \exp(-8k^2 \log n) = \frac{1}{n^{8k^2}}$$

And therefore, $k \leq \sqrt{\frac{C_1}{8}}$, which proves (a).

Since $n/2 + k\sqrt{n \log n}$ is an integer, then $f_0(x)$ equals one if and only if $L(x) \geq n/2 + k\sqrt{n \log n} + 1$. Therefore, we can rewrite Equation (D.1) as:

$$B[f_0] + \frac{1}{2^n} \cdot \binom{n}{n/2 + k\sqrt{n \log n}} > \frac{1}{n^{C_1}} \tag{D.2}$$

Additionally, for sufficiently large n we have:

$$B[f_0] = \frac{1}{2^n} \sum_{l=n/2+k\sqrt{n \log n}+1}^{n} \binom{n}{l} \geq \frac{1}{2^n} \cdot \binom{n}{n/2 + k\sqrt{n \log n} + 1}$$
$$= \frac{1}{2^n} \cdot \binom{n}{n/2 + k\sqrt{n \log n}} \cdot \frac{n - (n/2 + k\sqrt{n \log n} + 1) + 1}{n/2 + k\sqrt{n \log n} + 1}$$
$$= \left(1 - O\left(\sqrt{\frac{\log n}{n}}\right)\right) \cdot \frac{1}{2^n}\binom{n}{n/2 + k\sqrt{n \log n}} \geq \frac{1}{2} \cdot \frac{1}{2^n}\binom{n}{n/2 + k\sqrt{n \log n}} \tag{D.3}$$

Above, we used the fact that k is at most a constant. Combining Equations (D.2) and (D.3)

we get that for sufficiently large n:

$$3B[f_0] \geq \frac{1}{n^{C_1}} - \frac{1}{2^n}\binom{n}{n/2+k\sqrt{n\log n}} + 2 \cdot \frac{1}{2} \cdot \frac{1}{2^n}\binom{n}{n/2+k\sqrt{n\log n}} = \frac{1}{n^{C_1}}$$

This together with the fact that $B[f_0] \leq 1/n^{C_1}$, proves (b).

Consider (c) now. We have that:

$$\frac{1}{3n^{C_1}} \leq B[f_0] = \frac{1}{2^n} \sum_{l=n/2+k\sqrt{n\log n}+1}^{n} \binom{n}{l} \leq \frac{n}{2^n}\binom{n}{n/2+k\sqrt{n\log n}+1}$$

This implies that:

$$\Pr_{x\in_R\{0,1\}^n}[L(x) = n/2+k\sqrt{n\log n}+1] = \frac{1}{2^n}\binom{n}{n/2+k\sqrt{n\log n}+1} \geq \frac{1}{3n^{C_1+1}}$$

At the same time, given that $L(x) = n/2+k\sqrt{n\log n}+1$, if one flips an index i for which $x_i = 1$, then it will result that $f_0(x^{\oplus i}) = 0$. And since the number of such indices is at least half:

$$I[f_0] = n \cdot \Pr_{x\in_R\{0,1\}^n; i\in_R[n]}[f(x) \neq f(x^{\oplus i})] \geq$$
$$n \cdot \Pr_{x\in_R\{0,1\}^n}[L(x) = n/2+k\sqrt{n\log n}+1] \cdot \frac{1}{2} \geq n \cdot \frac{1}{3n^{C_1+1}} \cdot \frac{1}{2} = \Omega\left(\frac{1}{n^{C_1}}\right)$$

This proves the left inequality in (c). The right inequality is also correct, because it follows from Lemma 4.0.1 by picking F to be the all-zeros function. Thus, (c) is true.

Regarding noise sensitivity, a known lemma (stated in Appendix A as Lemma A.0.2) implies that noise sensitivity is an increasing function of δ. Therefore, it is enough to consider $\delta = 1/n$. Then, for any x, if we flip each index with probability $1/n$, the probability that overall exactly one index will be flipped equals $n \cdot \frac{1}{n}(1-1/n)^{n-1} = \Omega(1)$. Additionally, given that only one index is flipped, it is equally likely to be any of the n indices.

Therefore, we can lower-bound the noise sensitivity:

$$NS_\delta[f_0] \geq NS_{1/n}[f_0] = \mathrm{Pr}_{(x,y)\in_R T_{1/n}}[f_0(x) \neq f_0(y)]$$
$$\geq \Omega(1) \cdot \mathrm{Pr}_{x\in_R\{0,1\}^n, i\in_R[n]}[f_0(x) \neq f_0(x^{\oplus i})] = \Omega(1) \cdot \frac{1}{n} \cdot I[f_0]$$

Together with (c), this implies the left inequality in (d).

Regarding the right inequality, it follows from Lemma 4.0.1 by picking F to be the all-zeros function. Thus, (d) is true.