Approximating the noise sensitivity of a monotone Boolean function

Ronitt Rubinfeld * Arsen Vasilyan†

April 16, 2019

Abstract

The noise sensitivity of a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is one of its fundamental properties. A function of a positive noise parameter $\delta$, it is denoted as $NS_\delta[f]$. Here we study the algorithmic problem of approximating it for monotone $f$, such that $NS_\delta[f] \geq 1/n^C$ for constant $C$, and where $\delta$ satisfies $1/n \leq \delta \leq 1/2$. For such $f$ and $\delta$, we give a randomized algorithm performing $O\left(\min\left(1, \sqrt{\frac{n}{\log^{1.5} n}}\right) \cdot \text{poly}\left(\frac{1}{\epsilon}\right)\right)$ queries and approximating $NS_\delta[f]$ to within a multiplicative factor of $(1 \pm \epsilon)$. Given the same constraints on $f$ and $\delta$, we also prove a lower bound of $\Omega\left(\frac{\min(1, \sqrt{n})}{NS_\delta[f]} \cdot n^\xi\right)$ on the query complexity of any algorithm that approximates $NS_\delta[f]$ to within any constant factor, where $\xi$ 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 previously unknown lower bounds on the query complexity of approximating other fundamental properties of Boolean functions: the total influence and the bias.

*CSAIL at MIT, and the Blavatnik School of Computer Science at Tel Aviv University, ronitt@csail.mit.edu. Ronitt Rubinfeld’s research was supported by NSF grants CCF-1650733, CCF-1733808, IIS-1741137 and CCF-1740751.
†CSAIL at MIT, vasilyan@mit.edu. Arsen Vasilyan’s research was supported by the EECS SuperUROP program, the MIT Summer UROP program and the DeFlorez Endowment Fund.
1 Introduction

Noise sensitivity is a property of any Boolean function \( f : \{0,1\}^n \rightarrow \{0,1\} \) defined as follows: First, pick \( x = \{x_1, \ldots, x_n\} \) uniformly at random from \( \{0,1\}^n \), then pick \( z \) by flipping each \( x_i \) with probability \( \delta \). Here \( \delta \), the noise parameter, is a given positive constant no greater than \( 1/2 \) (and at least \( 1/n \) in the interesting cases). The noise sensitivity of \( f \), denoted as \( NS_\delta[f] \), equals the probability that \( f(x) \neq f(z) \). This definition was first explicitly given by Benjamini, Kalai and Schramm in [3]. Noise sensitivity has been the focus of multiple papers: [3, 7, 8, 10, 11, 13, 18, 23]. It has been applied to learning theory [4, 7, 8, 10, 11, 13, 18, 23], 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].

In this work, we study the algorithmic question of approximating the noise sensitivity of a function to which we are given oracle access. It can be shown that standard sampling techniques require \( O\left(\frac{1}{NS_\delta[f]^2}\right) \) 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 \( \epsilon \) is a sufficiently small constant, \( \delta \) satisfies \( 1/n \leq \delta \leq 1/2 \) and \( NS_\delta[f] \) satisfies \( \Omega\left(\frac{1}{n}\right) \leq NS_\delta[f] \leq O(1) \). Therefore, we focus on estimating the noise sensitivity of monotone functions, which form an important subclass of Boolean functions.

Noise sensitivity is closely connected to the total influence (henceforth just influence) of a Boolean function [21, 22]. Influence, denoted by \( I[f] \), is defined as \( n \) times the probability that \( f(x) \neq f(x^{\oplus i}) \), where \( x \) is chosen uniformly at random, as for noise sensitivity, and \( x^{\oplus i} \) is formed from \( x \) by flipping a random index (this latter probability is sometimes referred to as the average sensitivity). Despite their close connections, 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 \( \frac{1}{8}(1 - (1 - 2\delta)^n) \) (such disparities also hold for monotone functions, see for example the discussion of influence and noise sensitivity of the majority function in [22]).

The approximation of parameters of monotone Boolean functions was previously studied by [25, 24], where they considered the question of how fast one can approximate the influence of a monotone function \( f \) given oracle access to \( f \). It was shown 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 \( \epsilon \) beats the standard sampling algorithm by a factor of \( \sqrt{n} \), ignoring logarithmic factors.

Noise sensitivity is also closely related to the notion of noise stability (see for instance [22]). The noise stability of a Boolean function \( f \) depends on a parameter \( \rho \) and is denoted by \( \text{Stab}_\rho[f] \). A well known formula connects it with noise sensitivity \( NS_\delta[f] = \frac{1}{2}(1 - \text{Stab}_{1-2\delta}[f]) \). This implies that by obtaining an approximation for \( NS_\delta[f] \), one also achieves an approximation for the gap between \( \text{Stab}_{1-2\delta}[f] \) and one.

1.1 Results

Our main algorithmic result is the following:

Theorem 1 Let \( \delta \) be a parameter satisfying:

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

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

Note that computing noise-sensitivity using standard sampling requires $O\left(\frac{1}{NS_\delta[f] \epsilon^2}\right)$ samples. Therefore, for a constant $\epsilon$, 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 quantities 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 to the function. Additionally, it runs in time polynomial in $n$.

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

Remark: The range of the parameter $\delta$ can be divided into two regions of interest. In the region $1/n \leq \delta \leq \frac{1}{(\sqrt{n} \log n)}$, 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 $\tilde{O}(e^{C_1 \log n/2})$. 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 $\tilde{O}(e^{C_1 \log n/2})$.

We define the bias of a Boolean function as $B[f] \overset{\text{def}}{=} Pr[f(x) = 1]$, 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 using standard sampling, one needs $O(1/(B[f] \epsilon^2))$ queries. We obtain a lower bound for approximating it similar to the previous theorem:

1Standard 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 $f(x) \neq f(z)$.

2In the following, we discuss how we build on [25]. We discuss the relation with [24] in Subsection 1.4.

3Here and everywhere else, to reliably distinguish means to distinguish with probability at least 2/3.
**Theorem 3** Suppose \( n \) is a sufficiently large integer, \( C_1 \) and \( C_2 \) are constants satisfying \( C_1 - 1 > C_2 \geq 0 \). Then given a monotone function \( f : \{0,1\}^n \to \{0,1\} \), one needs at least \( \Omega\left(\frac{n^{C_2}}{\sqrt{e^2C_1^2 \log n/2}}\right) \) 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** Suppose \( n \) is a sufficiently large integer, \( C_1 \) and \( C_2 \) are constants satisfying \( C_1 - 1 > C_2 \geq 0 \). Then given a monotone function \( f : \{0,1\}^n \to \{0,1\} \), one needs at least \( \Omega\left(\frac{n^{C_2}}{\sqrt{e^2C_1^2 \log n/2}}\right) \) 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] 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]. There are many possibilities for algorithmic bounds that were compatible with the lower bound in [25], 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 run-time and query complexity, which are presented in Section 3.

First of all, with \( x \) and \( z \) described as above, using a standard pairing argument we argue that \( NS_{\delta}[f] = 2 \cdot \Pr[f(x) = 1 \land f(z) = 0] \). In other words, we can focus only on the case when the value of the function flips from one to zero.

We introduce the descending-ascending view of noise sensitivity (described in Subsection 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 define:

---

**Process D**

- **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.
- **Phase 1**: go through all the indexes in \( S_1 \) in a random order, and flip each with probability \( \delta \). Form the descending path \( P_1 \) from all the intermediate results. Call the endpoint \( y \).
- **Phase 2**: start at \( y \), and flip each index in \( S_0 \) with probability \( \delta \). As before, all the intermediate results form an ascending path \( P_2 \), which ends in \( z \).

---
Thus, we have the following algorithm:

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 Subsection 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 $t_1 \overset{\text{def}}{=} \eta_1 \sqrt{n \log n}$ and $t_2 \overset{\text{def}}{=} n \delta(1 + 3 \eta_2 \log n)$, where $\eta_1$ and $\eta_2$ 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 $n/2 + t_1$. Again, most of the edges in the hypercube are in $M$.

The utility of this ascending-descending view is that given $\delta \leq 1/(\sqrt{n \log^2 n})$, 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 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 following algorithm:

---

4Our analysis requires that only $\delta \leq 1/(\sqrt{n \log n})$ as in the statement of Theorem 1; however, the analysis of the case $1/(\sqrt{n \log^2 n}) \leq \delta \leq 1/(\sqrt{n \log n})$ involves technical subtleties that we ignore here.
Algorithm \( \mathcal{A} \) (given oracle access to a monotone function \( f : \{0, 1\}^n \to \{0, 1\} \) and a parameter \( \epsilon \))

1. Assign \( w = \frac{\epsilon}{300 \eta_2} \sqrt{\frac{n}{\log n}} \)
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.
4. If \( f(x) = f(y) \) output \( \text{FAIL} \).
5. If \( f(x) \neq f(y) \) perform a binary search on the path \( P_1 \) and find an influential edge \( e_{in, f} \).
6. If \( e_{in, f} \in M \) return \( e_{in, f} \). Otherwise output \( \text{FAIL} \).

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 \( w_1 \) 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 as \( \Lambda_e \) 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. Specifically, the algorithm is the following (recall that we use \( L(x) \) to denote the Hamming weight of \( x \) which equals to the number of indices \( i \) on which \( x_i = 1 \), and we use the symbol \( \Lambda_e \) to denote the whole layer of edges that have the same endpoint levels as \( e \)):

Algorithm \( \mathcal{W} \) (given an edge \( e \) so \( v_1 < v_2 \))

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 with probability \( \delta \), flip each index that equals to one to zero. The intermediate results form a path \( P_1 \), and we call its endpoint \( y \).
3. If \( P_1 \) does not intersect \( \Lambda_e \) go to step 1.
4. Otherwise, output \( w_1 = L(x) - L(v_1) \) and \( w_2 = L(v_2) - L(y) \).

Recall that \( t_2 \) has a technical role and is defined to be equal \( n \delta (1 + 3 \eta_2 \log n) \), where \( \eta_2 \) 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 \( e \) itself. Additionally, given the endpoint of \( P_1 \), we sample the path \( P_2 \) just as in the process \( D \). Formally, we use the following algorithm:

---

\( 5 \)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 \( \Lambda_e \).
1. Use $W(e)$ 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 $\delta$.
5. Return $P_1, P_2, x, y$ and $z$.

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 as follows:

**Algorithm for estimating noise sensitivity.** (given oracle access to a monotone function $f$: $\{0, 1\}^n \rightarrow \{0, 1\}$)

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 $A$ from Lemma 3.4 repeatedly to successfully sample an edge $e$.
   - From Lemma 3.8 use the algorithm $B$, 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 $2\tilde{p}_A \tilde{p}_B$.

### 1.3 Lower bound techniques

We use the same technique to lower bound algorithms which approximate 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 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 $D^B_1$ and $D^B_2$ that are relatively hard to distinguish but have drastically different biases. To create them, we use a special monotone function $F^B$, which we will explain later how to obtain. We pick a function from $D^B_2$ 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 $l_0$, where $l_0$ is some fixed threshold.

We form $D^B_1$ 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 $D^B_1$ has a sufficiently small bias.

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 $D_B^1$ and $D_B^2$ are hard to reliably distinguish. We say that a function is “thin” if it equals to one on only a small fraction of the points on the level $l_0$. To prove the first lemma, we show that one could transform any adaptive algorithm for distinguishing $D_B^1$ from $D_B^2$ 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 $D_B^2$ from a function in $D_B^1$ 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, 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 $2\sqrt{n}/nC_2$ 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 $D_B^1$ and $D_B^2$ 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 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 $O\left(\frac{1}{n^{1/11}}\right)$ and also improve the required lower bound on $I[f]$ to be $I[f] \geq \exp(-c_1\epsilon^2n + c_2\log(n/\epsilon))$ for some constant $c_1$ and $c_2$ (it was $I[f] \geq 1/nC$ 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.
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 \( \Lambda_e \) 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 \( \Lambda_e \) is \( L(v_1)^n \). The concept of \( \Lambda_e \) 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 \( \Lambda_e \).

Definition 4 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. We call the 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 the 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 hypercube 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 (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 \( C_2 \overset{\text{def}}{=} \frac{1}{10\sqrt{C_1}} \), then for any \( \xi \) 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 \( 1 - \xi \leq \frac{l_1}{l_2} \leq 1 + \xi \)

Proof: See Appendix C, Subsection 8.1
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 $\{0,1\}^n \rightarrow \{0,1\}^n$. 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 $f$: $\{0,1\}^n \rightarrow \{0,1\}$, denoted by $I[f]$ is:

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

Where $x \oplus i$ is $x$ with its $i$-th bit flipped. 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 $\delta$ 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 $T_\delta$. Then we define the noise sensitivity of $f$ as:

$$NS_\delta[f] \overset{\text{def}}{=} \Pr_{(x,z) \in RT_\delta}[f(x) \neq f(z)]$$

**Observation 2.2** For every pair $x^0$ and $z^0$, the probability that for a pair $x, z$ drawn from $T_\delta$, it is the case that $(x, z) = (x^0, z^0)$, is equal to the probability that $(x, z) = (z^0, x^0)$. Therefore,

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

Hence:

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

2.1.3 Influence estimation.

To estimate the influence, standard sampling would require $O\left(\frac{n}{\sqrt{|E_I|}}\right)$ samples. However, from [23] we have:

**Theorem 5** There is an algorithm that approximates $I[f]$ to within a multiplicative factor of $(1 \pm \epsilon)$ for a monotone $f$: $\{0,1\}^n \rightarrow \{0,1\}$. 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 $O\left(\frac{\sqrt{n \log(n/\epsilon)}}{I[f]^{1/2}}\right)$ queries. Additionally, it runs in time polynomial in $n$.

---

6 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.
2.1.4 Bounds for $\epsilon$ and $I[f]$

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

**Observation 2.3** In this work (recalling that $\delta \geq 1/n$), we assume that $\epsilon \geq H \sqrt{n\delta \log^{1.5}(n)} \geq H \frac{\log^{1.5}n}{\sqrt{n}}$, for any constant $H$, which is without loss of generality for the following reason. The standard sampling algorithm can estimate the noise sensitivity in $O\left(\frac{1}{NS_\delta[f]}\epsilon^2\right)$ samples. For $\epsilon = O\left(\sqrt{n\delta \log^{1.5}(n)}\right)$, this is $O\left(\frac{\sqrt{n\delta \log^{1.5}(n)}}{NS_\delta[f]}\right)$ and already accomplishes the desired query complexity. Additionally, throughout the paper whenever we need it, we will without loss of generality assume that $\epsilon$ is smaller than a sufficiently small positive constant.

We will also need a known lower bound on influence:

**Observation 2.4** For any function $f: \{0, 1\}^n \to \{0, 1\}$ and $\delta \leq 1/2$ it is the case that:

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

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

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

3 An improved algorithm for small $\delta$

In this section we give an improved algorithm for small $\delta$, 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.

It will also be useful to view noise sensitivity in the context of the following process.

---

**Process $D$**

- 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.

- **Phase 1**: go through all the indexes in $S_1$ in a random order, and flip each with probability $\delta$. Form the descending path $P_1$ from all the intermediate results. Call the endpoint $y$.

- **Phase 2**: start at $y$, and flip each index in $S_0$ with probability $\delta$. As before, all the intermediate results form an ascending path $P_2$, which ends in $z$.

---

7We 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 $Pr_Y[x = 1]$. 

10
By inspection, \( x \) and \( z \) are distributed identically in \( D \) as in \( T_3 \). Therefore from Observation 2.2:

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

**Observation 3.1** Since the function is monotone, if \( f(x) = 1 \) and \( f(z) = 0 \), then it has to be that \( f(y) = 0 \).

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:

\[
Q_{l,w} \overset{\text{def}}{=} Pr_D[|P_1| \geq w | L(x) = l]
\]

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:

\[
t_1 \overset{\text{def}}{=} \eta_1 \sqrt{n \log n} \quad t_2 \overset{\text{def}}{=} n\delta(1 + 3\eta_2 \log n)
\]

Here \( \eta_1 \) and \( \eta_2 \) are large enough constants. For constant \( C \), taking \( \eta_1 = \sqrt{C} + 4 \) and \( \eta_2 = C + 2 \) is sufficient for our purposes.

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:

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

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

We define the following two bad events in context of \( D \). 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.

- \( E_1 \) 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 wanted two things from it. First of all, we wanted its probability to be upper-bounded easily. Secondly, we wanted it not to complicate the sampling of paths in Lemma 3.6. There exists a tension between these two requirements, and as a result the definition of \( E_1 \) is somewhat contrived.
3.1.3 Defining \( p_A, p_B \)

We define:

\[
p_A \overset{\text{def}}{=} Pr_D[f(x) = 1 \land f(y) = 0 \land \overline{E_1} \land \overline{E_2}] \quad p_B \overset{\text{def}}{=} Pr_D[f(z) = 0 \land f(x) = 1 \land f(y) = 0 \land \overline{E_1} \land \overline{E_2}]
\]

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), 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:

\[
Pr_D[f(x) = 1 \land f(z) = 0 \land \overline{E_1} \land \overline{E_2}] = Pr_D[f(x) = 1 \land f(y) = 0 \land \overline{E_1} \land \overline{E_2}] = p_{APB} \tag{1}
\]

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

\[
p_e \overset{\text{def}}{=} Pr_D[e \in P_1 \land \overline{E_1} \land \overline{E_2}] \quad q_e \overset{\text{def}}{=} Pr_D[f(x) = 1 \land f(y) = 0 \land \overline{E_1} \land \overline{E_2}] = Pr_D[f(x) = 0 \land e \in P_1 \land \overline{E_1} \land \overline{E_2}]
\]

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 (1) and the laws of conditional probabilities we can write:

\[
p_A = \sum_{e \in E_I \cap M} p_e \tag{2}
\]

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:

\[
p_{APB} = Pr_D[(f(x) = 1) \land (f(z) = 0) \land \overline{E_1} \land \overline{E_2}] = \sum_{e \in E_I \cap M} p_e q_e \tag{3}
\]

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 3.2** 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:

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

The second inequality is the Hoeffding bound, then we used Observations 2.4 and 2.3.
**Lemma 3.3** We proceed to prove that ignoring these bad events does not distort our estimate for $\text{NS}_\delta[f]$. It is the case that:

$$p_{AB} \leq \frac{1}{2} \text{NS}_\delta[f] \leq \left(1 + \frac{\epsilon}{5}\right)p_{AB}$$

**Proof:** See Appendix C, Subsection 8.2.

---

### 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 3.4 is similar to the algorithm in [25]. In principle, we could have carried out much of the analysis of the algorithm in Lemma 3.4 by referencing an equation in [25]. However, for subsequent lemmas, including Lemma 3.5, 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 3.4 here, in order to demonstrate how the Hypercube Continuity Lemma (Lemma 2.1) can be used to analyze random walks on the hypercube, before handling the more complicated subsequent lemmas, including Lemma 3.5.

**Lemma 3.4** 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 $M \cap E_I$:

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

*The probability that the algorithm succeeds is at least $\Omega\left(\frac{1}{\sqrt{n \log^{2.5} n}}\right)$. 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:** Through the standard repetition technique, the probability of error can be decreased to an arbitrarily small constant, at the cost of $O\left(\frac{\sqrt{n \log^{2.5} n}}{I[e]}\right)$ queries. Then, the run-time still stays polynomial in $n$, since $I[n] \geq 1/n^C$.

**Remark:** 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:** The algorithm is as follows:
Algorithm $\mathcal{A}$ (given oracle access to a monotone function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and a parameter $\epsilon$)

1. Assign $w = \frac{\epsilon}{3100n} \sqrt{n \log n}$
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.
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.

Note that $P_1$ is distributed symmetrically with respect to change of indices. Pick an arbitrary edge $e_0$ in $M \cap E_I$ and let $e_0 = (v_1, v_2)$, so $v_2 \leq v_1$. Recall that $\Lambda_{e_0}$ was the set of edges in the same “layer” of the hypercube and that $|\Lambda_{e_0}|$ equals $L(v_1) \left( \begin{array}{c} n \\ L(v_1) \end{array} \right)$.

For any $e' \in \Lambda_{e_0}$, $e_0$ and $e'$ are different only up to a permutation of indexes. 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 $\Lambda_{e_0}$, which implies that the events of $P_1$ crossing each of these edges are disjoint. Thus, we can write:

$$Pr_{\mathcal{A}}[e_0 \in P_1] = \frac{Pr_{\mathcal{A}}[P_1 \cap \Lambda_{e_0} \neq \emptyset]}{L(v_1) \left( \begin{array}{c} n \\ L(v_1) \end{array} \right)}$$

But $P_1$ will intersect $\Lambda_{e_0}$ 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 $\Lambda_{e_0}$.

$$Pr_{\mathcal{A}}[e_0 \in P_1] = \frac{\sum_{L(v_1) + w - 1} L(v_1) \left( \begin{array}{c} n \\ L(v_1) \end{array} \right)}{L(v_1) \left( \begin{array}{c} n \\ L(v_1) \end{array} \right)}$$

Observation 2.3 allows us to lower-bound $\epsilon$ and argue that for sufficiently large $n$ we have:

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

Since $e \in M$, it is the case that $\frac{w}{2} - t_1 \leq L(v_2) \leq L(v_1) \leq \frac{w}{2} + t_1$. This allows us to use Lemma 2.1 and deduce that $1 - \epsilon/310 \leq \left( \begin{array}{c} n \\ L(v_1) \end{array} \right) \leq 1 + \epsilon/310$. This and Equation (5) allow us to approximate $Pr_{\mathcal{A}}[e_0 \in P_1]$ in Equation (4) the following way:

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

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:

$$Pr_{e \in \mathcal{A}}[e = e_0] = Pr_{\mathcal{A}}[e_{inf} = e_0] = Pr_{\mathcal{A}}\left[e_0 \in P_1 \left| e' \in M \cap E_I \right. \left( e' \in P_1 \right) \right] = \frac{Pr_{\mathcal{A}}[e_0 \in P_1]}{Pr_{\mathcal{A}}\left[ \bigvee_{e' \in M \cap E_I} \left( e' \in P_1 \right) \right]}$$

14
Substituting Equation (7) into Equation (6) we get:

\[
\left(1 - \frac{\epsilon}{150}\right) \frac{w}{n^{2n-1}} \Pr_{\Lambda} \left[ \bigvee_{e_1 \in M \cap E_I} (e_1 \in P_1) \right] \leq \Pr_{e \in R \setminus A}[e = e_0] \leq \left(1 + \frac{\epsilon}{150}\right) \frac{w}{n^{2n-1}} \Pr_{\Lambda} \left[ \bigvee_{e_1 \in M \cap E_I} (e_1 \in P_1) \right]
\]

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:

\[
\frac{1 - \epsilon/150}{1 + \epsilon/150} \Pr_{e \in R \setminus A}[e = e_2] \leq \Pr_{e \in R \setminus A}[e = e_1] \leq \frac{1 + \epsilon/150}{1 - \epsilon/150} \Pr_{e \in R \setminus A}[e = e_2]
\]

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 (6) over all edges in \(M \cap E_I\) and get that the probability of success is at least \(\Theta\left(\frac{|E_I \cap M|}{n^{2n-1}}\right)\). Applying Lemma 3.2 and substituting \(w\), the inverse of the success probability is:

\[
O\left(\frac{n^{2n-1}}{|E_I \cap M|} \cdot \log n\right) = O\left(\frac{n}{I[f]w} \cdot \log n\right) = O\left(\frac{\log^{1.5}(n) \sqrt{n}}{I[f]\epsilon}\right)
\]

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 3.5** 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 \(v_2 \leq v_1\). We can use the same argument from symmetry as in the proof of Lemma 3.4. We get:

\[
p_e = Pr\left[\{e \in P_1\} \cap \overline{E_1} \cap \overline{E_2}\right] = Pr\left[\{e \in P_1\} \cap \overline{E_1}\right]
\]

\[
= Pr\left[\{e \in P_1\} \cap (L(x) - L(v_1) < t_2)\right] = \frac{Pr\left[\{P_1 \cap \Lambda_e \neq \emptyset\} \cap (L(x) - L(v_1) < t_2)\right]}{L(v_1)\left(\frac{n}{L(v_1)}\right)}
\]

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 3.4.

Recall that we defined:

$$Q_{t,w} \overset{\text{def}}{=} \Pr_D[|P_1| \geq w|L(x) = t]$$

This allows us to rewrite:

$$p_e = \sum_{i=1}^{t_2} \frac{\frac{1}{2^i} \binom{n}{L(v_1)+i-1} Q_{L(v_1)+i-1;i}}{L(v_1) \binom{n}{L(v_1)}} \tag{8}$$

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 2.3 implies that for any $H$ we can assume that $\sqrt{n}\delta \leq \frac{\epsilon}{H \log^{1+\eta} n}$ using which we deduce

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

Furthermore, we will need a bound on the binomial coefficients in Equation (8). By picking $H$ to be a large enough constant, this allows us to use Lemma 2.1 to bound the ratios of the binomial coefficients. For any $i$ between 1 and $t_2$ inclusive:

$$1 - \frac{\epsilon}{1300} \leq \frac{\binom{n}{L(v_1)+i-1}}{\binom{n}{L(v_1)}} \leq 1 + \frac{\epsilon}{1300} \tag{9}$$

Equation (5) is valid in this setting too. Substituting Equation (5) together with Equation (9) into Equation (8) we get:

$$\left(1 - \frac{\epsilon}{630}\right) \frac{1}{n 2^{n-1}} \sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i} \leq p_e \leq \left(1 + \frac{\epsilon}{630}\right) \frac{1}{n 2^{n-1}} \sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i} \tag{10}$$

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, $Q_{t,w} \leq Q_{t+1,w}$. This allows us to bound:

$$\sum_{i=1}^{t_2} Q_{L(v_1);i} \leq \sum_{i=1}^{t_2} Q_{L(v_1)+i-1;i} \leq \sum_{i=1}^{t_2} Q_{L(v_1)+t_2-1;i} \tag{11}$$

Then, using Observation 2.3

$$\sum_{i=1}^{t_2} Q_{L(v_1)+t_2-1;i} \leq \sum_{i=1}^{t_2} Q_{L(v_1)+t_2-1;i} = E_D[L(x) - L(y)|L(x) = L(v_1) + t_2 - 1]$$

$$= \delta(L(v_1) + t_2 - 1) \leq \delta(n/2 + t_1 + t_2) \leq \left(1 + \frac{\epsilon}{630}\right) \frac{n \delta}{2} \tag{12}$$

Now, we bound from the other side:

$$\sum_{i=1}^{t_2} Q_{L(v_1);i} = \sum_{i=1}^{t_2} Q_{L(v_1);i} - \sum_{i=t_2+1}^{t_2} Q_{L(v_1);i} \geq \sum_{i=1}^{t_2} Q_{L(v_1);i} - n \cdot Q_{L(v_1);t_2} \tag{13}$$
We bound the first term just as in Equation (12), and we bound the second one using a Chernoff bound:

\[ \sum_{i=1}^{n} Q_L(v_1)_i = \delta L(v_1) \geq \delta(n/2 - t_1) \geq \left(1 - \frac{\epsilon}{1300}\right) \frac{\delta n}{2} \] (14)

\[ n \cdot Q_L(v_1)_t \leq n \cdot \exp\left(-\frac{1}{3}n \cdot \delta \eta_2 \log n\right) \leq \frac{1}{n^\eta_2 - 1} \] (15)

Substituting Equations (14) and (15) into Equation (13) and using Observation 2.3 we get:

\[ \sum_{i=1}^{t_2} Q_L(v_1)_i \geq \left(1 - \frac{\epsilon}{1300}\right) \frac{\delta n}{2} - \frac{1}{n^\eta_2 - 1} \geq \left(1 - \frac{\epsilon}{630}\right) \frac{\delta n}{2} \] (16)

Substituting Equations (16) and (12) into Equation (11) we get:

\[ \left(1 - \frac{\epsilon}{630}\right) \frac{\delta n}{2} \leq \sum_{i=1}^{t_2} Q_L(v_1)_i \leq \left(1 + \frac{\epsilon}{630}\right) \frac{\delta n}{2} \]

Combining this with Equation (10) we deduce that:

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

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

While we will use Lemma 3.5 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 \( E_1 \) and \( E_2 \). 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 3.8 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 3.8 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 3.4 and 3.5. 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 $\Lambda_e$. Then, we use the Lemma 2.1 to answer questions about random walks hitting a given layer. This is handled in Lemma 3.8.

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 3.8.

**Lemma 3.6** There is an algorithm $W$ that takes as input an edge $e = (v_1, v_2)$ in $M \cap E_I$, so that $v_2 \leq v_1$, and samples two non-negative numbers $w_1$ and $w_2$, so that for any two non-negative $w_1'$ and $w_2'$:

$$
(1 - \frac{\epsilon}{70}) \Pr_W([w_1 = w_1'] \land (w_2 = w_2']) \\
\leq \Pr_D([(L(x) - L(v_1) = w_1') \land (L(v_2) - L(y) = w_2') \mid (e \in P_1) \land \overline{E_1} \land \overline{E_2}]
$$

where $\Pr_D$ is the distribution of edges of $\Lambda_e$. Therefore:

$$
\Pr_W([w_1 = w_1'] \land (w_2 = w_2')] \\
\leq (1 + \frac{\epsilon}{70}) \Pr_W([w_1 = w_1'] \land (w_2 = w_2']) \\
\leq (1 + \frac{\epsilon}{70}) \Pr_D([(L(x) - L(v_1) = w_1') \land (L(v_2) - L(y) = w_2') \mid (e \in P_1) \land (L(x) - L(v_1) < t_2)]
$$

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

**Remark:** the approximation guarantee here is similar to the one in Lemma 3.4. It guaranteed that the relative distance should be small point-wise. This guarantee is stronger than closeness in either $L_1$ and $L_\infty$ norms. We also employ an analogous approximation guarantee in Lemma 3.8.

**Proof:** Here is the algorithm (recall that we used the symbol $\Lambda_e$ to denote the whole layer of edges that have the same endpoint levels as $e$):

**Algorithm $W$ (given an edge $e \triangleq (v_1, v_2)$ so $v_1 \leq v_2$):**

1. Pick an integer $t$ 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 with probability $\delta$, flip each index that equals to one to zero. The intermediate results form a path $P_1$, and we call its endpoint $y$.
3. If $P_1$ does not intersect $\Lambda_e$ go to step 1.
4. Otherwise, output $w_1 = L(x) - L(v_1)$ and $w_2 = L(v_2) - L(y)$.

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:

$$
\Pr_D([(L(x) - L(v_1) = w_1') \land (L(v_2) - L(y) = w_2') \mid e \in P_1 \land \overline{E_1} \land \overline{E_2}]
$$

$$
= \Pr_D \left[ (L(x) - L(v_1) = w_1') \land (L(v_2) - L(y) = w_2') \mid e \in P_1 \land (L(x) - L(v_1) < t_2) \right]
$$

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 $\Lambda_e$. Therefore:

18
\[ Pr_D[(L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2)](e \in P_1) \land E_1 \land E_2] \]
\[ = \sum_{e' \in \Lambda_e} Pr_D \left[ (L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2) \bigg| e' \in P_1 \land (L(x) - L(v_1) < t_2) \right] \cdot \frac{1}{|\Lambda_e|} \]
\[ = \sum_{e' \in \Lambda_e} Pr_D \left[ L(x) - L(v_1) = w'_1 \land (L(v_2) - L(y) = w'_2) \bigg| e' \in P_1 \land (L(x) - L(v_1) < t_2) \right] \times Pr_D \left[ e' \in P_1 \bigg| P_1 \cap \Lambda_e \neq \emptyset \right] \]
\[ = Pr_D \left[ (L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2) \bigg| (P_1 \cap \Lambda_e \neq \emptyset) \land (L(x) - L(v_1) < t_2) \right] \]
\[ = Pr_D \left[ L(x) - L(v_1) = w'_1 \land (L(v_2) - L(y) = w'_2) \bigg| (P_1 \cap \Lambda_e \neq \emptyset) \land (L(x) - L(v_1) < t_2) \right] \]

**Observation 3.7** 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:
\[ Q_{t,w} = Pr_D[|P_1| \geq w|L(x) = l] = Pr_{\mathcal{W}(e)}[|P_1| \geq w|L(x) = l] \]

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:
\[ Pr_{\mathcal{W}(e)}[L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2)] = \frac{1}{t_2} Pr_{\mathcal{W}(e)}[L(v_2) - L(y) = w'_2|L(x) - L(v_1) = w'_1] \]
\[ = \frac{1}{t_2} Pr_D[L(v_2) - L(y) = w'_2|L(x) - L(v_1) = w'_1] \]
\[ = \frac{1}{t_2} Q_{L(v_1)+w'_1,w'_2+1} Pr_D \left[ L(v_2) - L(y) = w'_2 \bigg| (L(x) - L(v_1) = w'_1) \land (|P_1| \geq w'_2 + 1) \right] \] (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 \( Q_{t,w} \).

Building on the previous equality we have:
\begin{align*}
Pr_{\mathcal{W}(e)}[(w_1 = w'_1) \land (w_2 = w'_2)] &= Pr_{\mathcal{W}(e)}[(L(x) - L(v_1) = w'_1) \land (L(v_2) - y) = w'_2)] Pr_{\mathcal{W}}(P_1 \cap \Lambda_e \neq \emptyset) \\
&= Pr_{\mathcal{W}(e)}[(L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2) \land (P_1 \cap \Lambda_e \neq \emptyset)] Pr_{\mathcal{W}}(P_1 \cap \Lambda_e \neq \emptyset) \\
&= Pr_{\mathcal{W}(e)}[(L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2)] Pr_{\mathcal{W}}(P_1 \cap \Lambda_e \neq \emptyset) \\
&= \frac{Q_{L(v_1) + w'_1} Pr_D}{\sum_{i=1}^{t_2} Q_{L(v_1) + i - 1; i}} \left[ L(v_2) - L(y) = w'_2 \left( L(x) - L(v_1) + w'_1 \right) \land (|P_1| \geq w'_1 + 1) \right]
\end{align*}

(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 3.7 since \( w'_1 \geq 1 \) and \( w'_2 \geq 0 \), hence \( \Lambda_e \) starts above \( \Lambda_e \) 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 (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 \( \Lambda_e \). Finally, we canceled \( \frac{1}{t_2} \) from both the numerator and the denominator.

Back to Equation (18). Analogous to how we derived Equation (19), we have:

\begin{align*}
Pr_D[(L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2)] &= Pr_D[L(x) - L(v_1) = w'_1] \cdot Pr_D[L(v_2) - L(y) = w'_2, L(x) - L(v_1) = w'_1] \\
&= \frac{1}{2^n} \left( \binom{n}{L(v_1) + w'_1} Q_{L(v_1) + w'_1; w'_1 + 1} \cdot Pr_D \left[ L(v_2) - L(y) = w'_2 \left( L(x) - L(v_1) = w'_1 \right) \land (|P_1| \geq w'_1 + 1) \right] \right)
\end{align*}

In addition to the steps analogous to the ones involved in getting Equation (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 (20) from Equation (19) using Bayes rule, we get:

\begin{align*}
Pr_D \left[ (L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2) \left( P_1 \cap \Lambda_e \neq \emptyset \right) \land (L(x) - L(v_1) < t_2) \right] &= Pr_D[(L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2) \left( P_1 \cap \Lambda_e \neq \emptyset \right) \land (L(x) - L(v_1) < t_2)] \\
&= \frac{Pr_D[(L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2) \left( P_1 \cap \Lambda_e \neq \emptyset \right) \land (L(x) - L(v_1) < t_2)]}{Pr_D[(P_1 \cap \Lambda_e \neq \emptyset) \land (L(x) - L(v_1) < t_2)]} \\
&= \frac{Pr_D[(L(x) - L(v_1) = w'_1) \land (L(v_2) - L(y) = w'_2)]}{Pr_D[(P_1 \cap \Lambda_e \neq \emptyset) \land (L(x) - L(v_1) < t_2)]} \\
&= \frac{1}{2^n} \left( \binom{n}{L(v_1) + w'_1} Q_{L(v_1) + w'_1; w'_1 + 1} \cdot Pr_D \left[ L(v_2) - L(y) = w'_2 \left( L(x) - L(v_1) = w'_1 \right) \land (|P_1| \geq w'_1 + 1) \right] \right) \\
&= \frac{1}{2^n} \sum_{i=1}^{t_2} \frac{1}{2^n} \left( \binom{n}{L(v_1) + i - 1} Q_{L(v_1) + i - 1; i} \right)
\end{align*}

(21)
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 \([5, 7]\) they are implied by the first two clauses.

Since by Observation \([3, 7]\) it is the case that \(1 \leq i \leq t_2\) and \(0 \leq w_1' < t_2\), the same way we proved Equation \([6]\) we have:

\[
1 - \frac{\epsilon}{150} \leq \frac{\binom{n}{i} \binom{n}{i-1}}{\binom{n}{i+w_1'+1}} \leq 1 + \frac{\epsilon}{150}
\]  

(22)

Combining Equations \((21)\) and \((22)\) we get:

\[
\left(1 - \frac{\epsilon}{70}\right) Q_{L(v_1)+w_1'+1} \cdot Pr_{D}[L(v_2) - L(y) = w_2'] \left(\frac{L(x) - L(v_1) = w_1'}{\binom{n}{i+w_1'+1} \cdot \sum_{i=1}^{t_2} Q_{L(v_1)+i-1,i}}\right) \\
\leq Pr_{D}[L(x) - L(v_1) = w_1'] \wedge (L(v_2) - L(y) = w_2') \left(\frac{P_1 \cap \Lambda_e \neq \emptyset}{} \wedge (L(x) - L(v_1) \leq t_2)\right) \\
\leq \left(1 + \frac{\epsilon}{70}\right) Q_{L(v_1)+w_1'+1} \cdot Pr_{D}[L(v_2) - L(y) = w_2'] \left(\frac{L(x) - L(v_1) = w_1'}{\binom{n}{i+w_1'+1} \cdot \sum_{i=1}^{t_2} Q_{L(v_1)+i-1,i}}\right) \\
\]  

(23)

Substituting Equations \((18)\) and \((20)\) into Equation \((23)\) 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 \(\exp\left(-\frac{1/4\delta n/4}{2}\right) = \exp\left(-\frac{\delta}{16}\right) \leq \exp(-\frac{1}{32})\). Therefore:

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

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

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

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

\[\text{Lemma 3.8} \quad \text{There exists an algorithm } B \text{ with the following properties. It takes as input an edge } e = (v_1, v_2) \text{ in } M \cap E_1, \text{ so that } v_2 \leq v_1 \text{ and outputs paths } P_1 \text{ and } P_2 \text{ together with hypercube vertices } x, y \text{ and } z. \text{ It is the case that } x \text{ is the starting vertex of } P_1, y \text{ is both the starting vertex of } P_2 \text{ and the last vertex of } P_1, \text{ and } z \text{ is the last vertex of } P_2. \text{ Additionally, } P_1 \text{ is descending and } P_2 \text{ is ascending. Furthermore, for any pair of paths } P_1' \text{ and } P_2' \text{ we have:}
\]

\[
|Pr_{B(e)}[(P_1 = P_1') \wedge (P_2 = P_2')] - Pr_{D}[(P_1 = P_1') \wedge (P_2 = P_2')]|(e \in P_1) \wedge (e \in P_2)\lvert(\overline{E_1} \wedge E_2)\rvert \\
\leq \frac{\epsilon}{70} \cdot Pr_{B(e)}[(P_1 = P_1') \wedge (P_2 = P_2')] \\
\]

(24)

It requires no queries to the function and takes computation time polynomial in \(n\) to draw one sample.
Similarly, we can write:

\[
Pr_D \left[ (P_1 = P'_1) \land (P_2 = P'_2) \right] (e \in P_1) \land \overline{E}_1 \land \overline{E}_2
\]

\[
= Pr_D \left[ (L(x) = L(x')) \land (L(y) = L(y')) \right] (e \in P_1) \land \overline{E}_1 \land \overline{E}_2
\]

\[
\times Pr_D \left[ (P_1 = P'_1) \land (P_2 = P'_2) \right] (e \in P_1) \land \overline{E}_1 \land \overline{E}_2 \land (L(x) = L(x')) \land (L(y) = L(y')) \right] \tag{25}
\]

Similarly, we can write:

\[
Pr_{B(e)}[(P_1 = P'_1) \land (P_2 = P'_2)] = Pr_{B(e)}[(L(x) = L(x')) \land (L(y) = L(y'))]
\]

\[
\times Pr_{B(e)}[L(x) = L(x')) \land (L(y) = L(y'))] \tag{26}
\]

By Lemma 3.6, we know that:

\[
\left(1 - \frac{\epsilon}{70}\right) Pr_{B(e)}[(L(x) = L(x')) \land (L(y) = L(y'))]
\]

\[
\leq Pr_D \left[ (L(x) = L(x')) \land (L(y) = L(y')) \right] (e \in P_1) \land \overline{E}_1 \land \overline{E}_2
\]

\[
\leq \left(1 + \frac{\epsilon}{70}\right) Pr_{B(e)}[(L(x) = L(x')) \land (L(y) = L(y'))] \tag{27}
\]

Considering Equations (25), (26) and (27) together, for the lemma to be true, it is enough that:

\[
Pr_D[(P_1 = P'_1) \land (P_2 = P'_2)](e \in P_1) \land \overline{E}_1 \land \overline{E}_2 \land (L(x) = L(x')) \land (L(y) = L(y'))
\]

\[
= Pr_{B(e)}[(P_1 = P'_1) \land (P_2 = P'_2)](L(x) = L(x')) \land (L(y) = L(y')) \tag{28}
\]

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 (28) is equivalent to:
Using the algorithm from [25] as described in Theorem 5, compute an approximation to the influence
Equation (30) we get Equation (29) which completes the proof of correctness.

This comes from the fact that in both $D$ and $B(e)$ the distribution of $P_1$ is symmetric with respect to exchange
of coordinates. Now we argue that:

\[ Pr_D[P_1 \neq P'_1] \land (L(x) = L(x')) \land (L(y) = L(y'))] = Pr_{B(e)}[P_1 \neq P'_1](L(x) = L(x')) \land (L(y) = L(y'))] \quad (30) \]

This is true because given $P_1$, both $D$ and $B(e)$ choose $P_2$ the same way. Combining Equation (31) with
Equation (30) we get Equation (29) which completes the proof of correctness.

The claims about run-time follow from the run-time guarantee on $W$. ■

3.5 The noise sensitivity estimation algorithm.

Our algorithm is:

**Algorithm for estimating noise sensitivity.** (given oracle access to a monotone function $f: \{0, 1\}^n \rightarrow \{0, 1\}$)

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 $I$. 
2. Compute $\hat{p}_A := \delta I / 2$.
3. Initialize $\alpha := 0$ and $\beta := 0$. Repeat the following until $\alpha := 768 \ln 200 / \epsilon^2$.
   • Use algorithm $A$ from Lemma 3.4 repeatedly to successfully sample an edge $e$.
   • From Lemma 3.8 use the algorithm $B$, 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 $\hat{p}_B = \frac{\beta}{\beta}$.
5. Return $2p_A \hat{p}_B$.

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

**Lemma 3.9** 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 3.5 over all the edges in $E_I \cap M$ we get that:

\[ \left(1 - \frac{\epsilon}{310}\right) \delta \cdot |E_I \cap M| / 2^n \leq \sum_{e \in E_I \cap M} p_e \leq \left(1 + \frac{\epsilon}{310}\right) \delta \cdot |E_I \cap M| / 2^n \]
Substituting Equation (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 3.2 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 3.10** 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 $\phi$ 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 $\phi$ and that $\phi$, 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 3.11** 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, Subsection 8.3

**Lemma 3.12** 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, Subsection 8.4

Corollary 3.10 with Lemmas 3.11 and 3.12 together imply that with probability at least 2/3:
\[
\left(1 - \frac{\epsilon}{5}\right) \hat{p}_{A} \hat{p}_{B} \leq p_{A} p_{B} \leq \left(1 + \frac{\epsilon}{5}\right) \hat{p}_{A} \hat{p}_{B}
\]

Combining this with Lemma 3.3 and Equation (3) we get that:

\[
\left(1 - \frac{\epsilon}{2}\right) 2\hat{p}_{A} \hat{p}_{B} \leq NS_{\delta}[f] \leq \left(1 + \frac{\epsilon}{2}\right) 2\hat{p}_{A} \hat{p}_{B}
\]

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

- Estimating the influence requires \(O\left(\sqrt{n \log(n/\epsilon)}\right)\) queries and polynomial time. Since by Observation 2.3 it is the case that \(\epsilon \geq 1/n\), this is at most \(O\left(\sqrt{n \log(n)}\right)\).

- By Lemma 3.4, successfully sampling an edge requires \(O\left(\sqrt{n \log(1.5 n)}\right)\) queries and polynomial time. By Lemma 3.8 for each edge we will additionally spend a polynomial amount of extra time.

- By Lemmas 3.11 and 3.12 we will have \(O\left(\frac{1}{p_{B} \epsilon^{2}}\right)\) iterations of the loop.

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

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

Now using Lemmas 3.9 and 3.3 together with Equation (3) it equals to:

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

Which is the desired number of queries.

# 4 Lower bounding the sample complexity.

We begin our proof of Theorems 2, 3 and 4 by first defining the distributions \(D_{1}^{\uparrow}, D_{1}^{\downarrow}\) and \(D_{1}^{\delta}\) to consist of a single function \(f_{0}: \{0, 1\}^{n} \rightarrow \{0, 1\}\):

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

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 4.1** For any monotone function \(F: \{0, 1\}^{n} \rightarrow \{0, 1\}\), 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) \[ |I[F] - I[F']| \leq \frac{2n}{n^{C_1}} \]

c) For any \( \delta \):
\[ |NS_\delta[F] - NS_\delta[F']| \leq \frac{2}{n^{C_1}} \]

**Proof:** See Appendix D, Subsection 9.1.

**Lemma 4.2** 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 \( \delta \), 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, Subsection 9.2.

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 4.3** Suppose \( F: \{0, 1\}^n \to \{0, 1\} \) is a monotone Boolean function with the property that:

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

Additionally, suppose \( C \) 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 \( \sigma \) is a random permutation of indices.

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

**Lemma 4.4** 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 \( F^\delta: \{0, 1\}^n \to \{0, 1\} \) such that:
• Any $f$ in $\{F^B, F^I\} \cup \{F^\delta : 1/n \leq \delta \leq 1/2\}$ has the property that:
\[ Pr_{x \in \{0,1\}^n}[f(x) = 1|L(x) = n/2 + k\sqrt{n\log n}] \leq \Theta \left( 1/n^{C_2} \cdot e^{C_1 \log n/2} \right) \]
• $B[F^B] \geq \Omega(1/n^{C_2})$.
• $I[F^I] \geq \Omega(\sqrt{n}/n^{C_2})$.
• 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 $D^B_1$, $D^I_1$ and $D^\delta_1$ 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 4.3.

• $D^B_2$ as the OR of $f_0(x)$ and $F^B(\sigma(x))$. Where, recall, $\sigma$ is a random permutation of indices.
• $D^I_2$ as the OR of $f_0(x)$ and $F^I(\sigma(x))$.
• For each $1/n \leq \delta \leq 1/2$, we define $D^\delta_2$ as the OR of $f_0(x)$ and $F^\delta(\sigma(x))$.

Observation 4.5 Permuting the indices to an input of a function preserves its bias, influence and noise sensitivity. That, together with Lemma 4.4 and Lemma 4.3 implies that:

a) For any $f$ in $D^B_2$, we have $B[f] \geq \Omega(1/n^{C_2})$.

b) For any $f$ in $D^I_2$, we have $I[f] = \Omega(\sqrt{n}/n^{C_2}) - O(n/n^{C_1}) = \Omega(\sqrt{n}/n^{C_2})$. 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 $D^\delta_2$:
\[ NS_\delta[f] \geq \begin{cases} 
\Omega(\delta\sqrt{n}/n^{C_2}) - O(1/n^{C_1}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\
\Omega(1/n^{C_2}) - O(1/n^{C_1}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 
\end{cases} = \begin{cases} 
\Omega(\delta\sqrt{n}/n^{C_2}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\
\Omega(1/n^{C_2}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 
\end{cases} \]

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 4.3 prevents us from distinguishing. Suppose $L_B$ is an algorithm, taking as input a monotone function $f : \{0,1\}^n \rightarrow \{0,1\}$ 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 $o \left( \frac{n^{C_2}}{e^C_1 \log n/2} \right)$ queries.

By Lemma 4.2, every function in $D_1$ has a bias of $\Theta(1/n^{C_1})$ and by Observation 4.5 the bias of every function in $D^B_2$ is at least $\Omega(1/n^{C_2})$. Therefore, $L_B$ can distinguish between them with probability at least 2/3 making $o \left( \frac{n^{C_2}}{e^C_1 \log n/2} \right)$. But by Lemma 4.3 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 4.3

Suppose $C$ 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:

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

We call the difference above the distinguishing power of $C$.

**Observation 4.6** 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 $C$ never queries any point in that region.

**Observation 4.7** If $C$ is randomized, we can think of it as a probability distribution over deterministic algorithms. The distinguishing powers of $C$ 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 $C$. Thus, without loss of generality we can assume that $C$ is deterministic.

Now, since $C$ is deterministic and it makes $q$ queries, it can be represented as a decision tree of depth $q$. At each node, $C$ 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 4.6 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 $p_{\text{get1}}$.

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 $C$. Then, the distinguishing power $p_C$ 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 4.8** There exists an algorithm $D$ that satisfies all of the following:
• It is deterministic and non-adaptive.
• Its distinguishing power between $D_1$ and $D_2$ is at least $p_C$.
• Just as $C$ it makes $q$ queries. We call them $z^1, \ldots, z^q$.
• 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 $C$ makes along the path $A$. Call them $y^1, \ldots, y^q$. 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 $D$ 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, $D$ has a distinguishing power of at least $\frac{1}{2}(1 + p_{get1})$. This implies that the distinguishing power of $D$ is at least that of $C$.

Now, we can bound the distinguishing power of $D$, which we call $p_D$. We have:

$$\frac{1}{2} Pr[D \text{ returns } 2| f \in R D_2] = Pr_{f \in R D_2} \left[ \bigvee_{j=0}^{q} f(z^j) = 1 \right] \leq \sum_{j=0}^{q} Pr_{f \in R D_2} [f(z^j) = 1]$$

$$= \sum_{j=0}^{q} Pr_{\sigma \text{ is a random permutation}} \left[ F(\sigma(z^j)) = 1 \right] = q \cdot Pr_{x \in \{0,1\}^n} \left[ F(x) = 1 \right] \frac{L(x) = n/2 + k\sqrt{n\log n}}{q_{0}} \leq \frac{q}{q_0}$$

Above we used (i) a union bound and the fact that by Lemma 4.8 the algorithm $D$ is non-adaptive (ii) The fact that by definition $f(x) = F(\sigma(x))$. Recall that $\sigma$ is the random permutation $F$ was permuted with. (iii) For any constant $x$, $\sigma(x)$ is uniformly distributed among the vertices with the same level. (iv) Lemma 4.8 together with the condition on the function $F$.

Therefore, we get that:

$$p_D \overset{def}{=} \frac{1}{2} Pr[D \text{ returns } 2| f \in R D_2] + \frac{1}{2} Pr[D \text{ returns } 1| f \in R D_1] \leq \frac{q}{q_0} + \frac{1}{2}$$

Since $q = o(q_0)$, then $p_D$ has to be less than $2/3$. Since $p_C$ is at most $p_D$ by Lemma 4.8 then $p_C$ also has to be less than $2/3$. This proves the lemma.

4.2 Proof of Lemma 4.4

Consider the distribution $H$ 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:

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.
a) Any \( f \) in \( H \) has the property that:

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

b) \( E_{f \in RH}[B[f]] \geq \Omega(1/n^{C_2}) \).

c) \( E_{f \in RH}[I[f]] \geq \Omega(\sqrt{n}/n^{C_2}) \).

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

\[
E_{f \in RH}[\mathcal{NS}_\delta[f]] \geq \begin{cases} 
\Omega(\delta \sqrt{n}/n^{C_2}) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\
\Omega(1/n^{C_2}) & \text{if } 1/\sqrt{n} < \delta \leq 1/2
\end{cases}
\]

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:

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

In the very end we used that by Lemma 4.2 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 \( 1/n^{C_2} \cdot 2^{\sqrt{n}} \) using a union bound. This gives us an upper bound of \( \Theta \left( \frac{1}{n^{C_2}} \cdot e^{C_1 \log n/2} \right) \).

### 4.4 Proof of b)

It is the case that:

\[
E_{f \in RH}[B[f]] = E_{f \in RH}[E_{x \in \{0,1\}^n}[f(x)]] = E_{x \in \{0,1\}^n}[E_{f \in RH}[f(x)]] \geq \frac{1}{2} E_{x \in \{0,1\}^n} \left[ E_{f \in RH}[f(x)] \left| L(x) \geq \frac{n}{2} \right. \right]
\]

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:
\[ \frac{L(x)}{n} \cdot \frac{L(x) - 1}{n - 1} \cdots \frac{L(x) - \sqrt{n} + 1}{n - \sqrt{n} + 1} \geq \left( \frac{L(x) - \sqrt{n} + 1}{n - \sqrt{n} + 1} \right)^\sqrt{n} \geq \left( \frac{n/2 - \sqrt{n} + 1}{n - \sqrt{n} + 1} \right)^\sqrt{n} \geq \left( \frac{1}{2} - \frac{\sqrt{n}/2 - 1/2}{n - \sqrt{n} + 1} \right)^\sqrt{n} \geq \frac{1}{2\sqrt{n}} \left( 1 - \frac{1}{\Theta(\sqrt{n})} \right)^\sqrt{n} = \Theta(2^{-\sqrt{n}}) \]

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

\[ E_{x \in R(0,1)^n} \left[ E_{f \in R_H} [f(x) \mid L(x) \geq n/2] \right] \geq 1 - \left( 1 - \frac{1}{\Theta(2^{-\sqrt{n}})} \right)^{2\sqrt{n}/n^{C_2}} \geq \Theta(1/n^{C_2}) \]

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

### 4.5 Proof of c)

We have that:

\[ E_{f \in R_H} [I[f]] = E_{f \in R_H} \left[ n \cdot Pr_{x \in R(0,1)^n,i \in R[n]} [f(x) \neq f(x^{(i)})] \right] = n \cdot Pr_{f \in R_H,x \in R(0,1)^n,i \in R[n]} [f(x) \neq f(x^{(i)})] \]

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 \(32\) it follows:

\[ E_{f \in R_H} [I[f]] \geq n \cdot Pr_{f \in R_H,x \in R(0,1)^n,i \in R[n]} \left[ f(x) \neq f(x^{(i)}) \mid n/2 - \sqrt{n} \leq L(x) \leq n/2 + \sqrt{n} \right] \times Pr_{x \in R(0,1)^n} [n/2 - \sqrt{n} \leq L(x) \leq n/2 + \sqrt{n}] \]

\[ \geq 0.95 n \cdot Pr_{f \in R_H,x \in R(0,1)^n,i \in R[n]} \left[ f(x) \neq f(x^{(i)}) \mid n/2 - \sqrt{n} \leq L(x) \leq n/2 + \sqrt{n} \right] \]

(33)

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

\[ Pr_{f \in R_H} [\wedge_j \text{ is satisfied}] = \frac{L(x)}{n} \cdot \frac{L(x) - 1}{n - 1} \cdots \frac{L(x) - \sqrt{n} + 1}{n - \sqrt{n} + 1} \]

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:

\[ Pr_{f \in R_H} [\wedge_j \text{ is satisfied}] = \frac{1}{\Theta(2^{\sqrt{n}})} \]

(34)

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 $\land_i$. By Equation (34), the probability of this happening is $1/\Theta(2\sqrt{n})$.

• $x$ does not satisfy all the other $\frac{1}{n^{c_2}}2\sqrt{n}$ clauses. Since they are chosen independently, by Equation (34) 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 $\land_i$. The probability of this is $1/\sqrt{n}$.

Since these three events are independent:

$$P_{f \in RH} [M_i] \geq \frac{1}{\Theta(2\sqrt{n})} \cdot \Theta(1) \cdot \frac{1}{\sqrt{n}} = \frac{1}{\Theta(\sqrt{n}2\sqrt{n})}$$

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:

$$P_{f \in RH, i \in R[n]} [f(x) \neq f(x^{\oplus i})] \geq P_{f \in RH, i \in R[n]} \left[ \bigvee_i M_i \right] \geq \frac{2\sqrt{n}}{n^{c_2}} \cdot \frac{1}{\Theta(\sqrt{n}2\sqrt{n})} = \frac{1}{\Theta(\sqrt{n} \cdot n^{c_2})}$$

(35)

Combining Equations (33) and (35):

$$E_{f \in RH} [I[f]] \geq 0.95n \cdot \frac{1}{\Theta(\sqrt{n} \cdot n^{c_2})} = \Omega\left(\frac{\sqrt{n}}{n^{c_2}}\right)$$

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 $\delta$. We have:

$$E_{f \in RH} [NS_\delta[f]] = E_{f \in RH} \left[ P_{(x,y) \in RT_\delta} [f(x) \neq f(y)] \right] = P_{f \in RH, (x,y) \in RT_\delta} [f(x) \neq f(y)]$$

(36)

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 $1 - \exp(-\delta n/8) \geq 1 - \exp(-1/8) \geq 0.11$. 

32
By a union bound, with probability at least 0.01 all the events $G_1, G_2$ and $G_3$ happen. Therefore:

$$E_{f \in RH}[NS_0[f]] \geq Pr_{f \in RH,(x,y) \in RT_3} \left[ f(x) \neq f(y) \left| G_1 \land G_2 \land G_3 \right. \right] \cdot Pr_{(x,y) \in RT_3}[G_1 \land G_2 \land G_3]$$

$$\geq 0.01 \cdot Pr_{f \in RH,(x,y) \in RT_3} \left[ f(x) \neq f(y) \left| G_1 \land G_2 \land G_3 \right. \right] \geq 0.01 \cdot Pr_{f \in RH}[f(x) \neq f(y)] \geq 0.01 \cdot Pr_{f \in RH}[f(x) \neq f(y)]$$

(37)

Now, suppose we are given values of $x$ and $y$ that satisfy $G_1 \land G_2 \land G_3$, we will lower bound $Pr_{f \in RH}[f(x) \neq f(y)]$. Using $G_1$ and $G_2$, just as in the proof of equation (5), we have that for any clause $\land j$:

$$Pr_{f \in RH}[\land j \text{ is satisfied by } x] = \frac{1}{\Theta(2^{\sqrt{n}})} \quad Pr_{f \in RH}[\land j \text{ is satisfied by } y] = \frac{1}{\Theta(2^{\sqrt{n}})}$$

(38)

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 $\land i$. By Equation (38), the probability of this is at least $1/\Theta(2^{\sqrt{n}})$.
- All the $\land j$ for $j \neq i$ are unsatisfied by both $x$ and $y$. By Equation (38) 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 $\land i$, it happens that at least one of the coordinates relevant to $\land i$ is zero in $y$. We call the probability of this happening $p_{\text{flip}}$.

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 $\land i$, 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:

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

(39)

We now lower-bound $p_{\text{flip}}$. Because of $G_3$, at least $\delta|S_x|/2$ of the indices in $S_x$ become zero in $y$. Let $S_{\land i}$ be the set of $\sqrt{n}$ indices relevant to $\land i$. Since they were chosen uniformly at random then, conditioning on $\land i$ being satisfied, $S_{\land i}$ 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 $y$:

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

Combining this with Equation (39), we get:

$$Pr_{f \in RH}[N_i] \geq \frac{p_{\text{flip}}}{\Theta(2^{\sqrt{n}})} \begin{cases} \Theta(\delta \sqrt{n}/2^{\sqrt{n}}) \quad \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Theta(1/2^{\sqrt{n}}) \quad \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases}$$

(40)

33
We now combine (i) Equation (37) (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 (40):

$$E_{f \in R}[NS_\delta[f]] \geq 0.01 \cdot Pr_{f \in R}[\bigvee_i N_i] = 0.01 \cdot \frac{2\sqrt{\pi}}{n^{c_2}} Pr_{f \in R,H,i \in [n]}[N_i] \geq \begin{cases} \Theta(\delta \sqrt{n}/nc_2) & \text{if } 1/n \leq \delta \leq 1/\sqrt{n} \\ \Theta(1/nc_2) & \text{if } 1/\sqrt{n} < \delta \leq 1/2 \end{cases}$$

5 Acknowledgments

We are grateful to the anonymous referees for helpful comments and suggestions.

References

[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 ODonnell, 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.
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]:

6 Appendix A

In this section we use Fourier analysis of boolean functions. We will use the notation of [22].
Lemma 6.1  For any function \( f: \{0, 1\}^n \rightarrow \{0, 1\} \) 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]):

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

We can now use Bernoulli’s inequality \((1 - |S| \leq 1 - 2|S|)\). Therefore:

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

This completes the proof.  ■

Lemma 6.2  For a fixed function \( f: \{0, 1\}^n \rightarrow \{0, 1\} \) and for values of \( \delta \) satisfying \( 0 < \delta \leq 1/2 \), \( NS_\delta[f] \) is an increasing function of \( \delta \).

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

7 Appendix B

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

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

Lemma 7.1  For any sufficiently large \( n \), any \( \alpha \), satisfying \( 10^n/2^n \leq \alpha \leq 1/2 \) and any \( \delta \), 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:

\[
E_{f \in R^H^\alpha}[NS_\delta[f]] = Pr_{f \in R^H^\alpha; (x,y) \in R^2 \delta}[f(x) \neq f(y)] = Pr_{(x,y) \in R^2 \delta}[x \neq y] \cdot Pr_{f \in R^H^\alpha; (x,y) \in R^2 \delta}[f(x) \neq f(y)|x \neq y]
\]

(41)

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

\[
Pr_{(x,y) \in R^2 \delta}[x \neq y] = 1 - (1 - \delta)^n \geq 1 - (1 - 1/n)^n \geq 1 - e^{-1} \geq 0.25 \quad (42)
\]

Additionally, we have:

\[
Pr_{f \in R^H^\alpha; (x,y) \in R^2 \delta}[f(x) \neq f(y)|x \neq y] = 2\alpha \cdot \frac{2^n \cdot (1 - \alpha)}{2^n - 1} \quad (43)
\]

Combing equations (41), (7) and (43), we get:
\begin{align*}
E_{f \in R^x} [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{44}
\end{align*}
At the same time, for sufficiently large \( n \) we have that:
\begin{align*}
E_{f \in R^x} [NS_\delta[f]] &= Pr_{(x,y) \in R} T_3 [x \neq y] \cdot 2\alpha \cdot \frac{2^n \cdot (1 - \alpha)}{2^n - 1} \leq 2.5\alpha \tag{45}
\end{align*}
Now, we will bound the variance. We have:
\begin{align*}
E_{f \in R^x} [(NS_\delta[f])^2] &= Pr_{f \in R^x} [(f(x^1) \neq f(y^1)) \land (f(x^2) \neq f(y^2))] \\
&= (Pr_{(x,y) \in R} T_3 [x \neq y])^2 \cdot Pr_{f \in R^x} [(f(x^1) \neq f(y^1)) \land (f(x^2) \neq f(y^2))] (x^1 \neq y^1) \land (x^2 \neq y^2) \tag{46}
\end{align*}
We have the following three facts:
1. For arbitrary \( x^1, y^1, x^2 \) and \( y^2 \), we have
\[ Pr_{f \in R^x} [(f(x^1) \neq f(y^1)) \land (f(x^2) \neq f(y^2))] \leq Pr_{f \in R^x} [(f(x^1) \neq f(y^1))] \leq 2\alpha \cdot 2^n (1 - \alpha) / (2^n - 1) \]
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
\[ Pr_{f \in R^x} [(f(x^1) \neq f(y^1)) \land (f(x^2) \neq f(y^2))] = 4\alpha \cdot \frac{2^n (1 - \alpha)}{2^n - 1} \cdot \frac{2^n - 1}{2^n - 2} \cdot \frac{2^n (1 - \alpha) - 1}{2^n - 3} \]
3. If \( (x^1, y^1) \) and \( (x^2, y^2) \) are picked independently from \( T_\delta \) conditioned on \( x^1 \neq y^1 \) and \( x^2 \neq y^2 \), then \( K \) happens unless \( x^1 = x^2 \) or \( x^1 = y^2 \) and \( x^2 = y^1 \). By independence and a union bound, the probability of any of these happening is at most \( 4/2^n \).

Therefore:
\begin{align*}
&Pr_{f \in R^x} [(f(x^1) \neq f(y^1)) \land (f(x^2) \neq f(y^2))] (x^1 \neq y^1) \land (x^2 \neq y^2) \\
&\leq Pr_{f \in R^x} [(f(x^1) \neq f(y^1)) \land (f(x^2) \neq f(y^2))] (x^1 \neq y^1) \land (x^1 \neq y^2) \land (y^1 \neq y^2) \\
&\lor (y^1 = x^2) \lor (y^1 = y^2) \cdot Pr_{(x,y) \in R} T_3 [(x^1 = x^2) \lor (x^1 = y^1) \lor (y^1 = x^2) \lor (y^1 = y^2)] \tag{47}
\end{align*}
Combining this with (44) and (46) we get a bound for the variance:
\begin{align*}
Var_{f \in R^x} [NS_\delta[f]] &= (Pr_{(x,y) \in R} T_3 [x \neq y])^2 \cdot \left( 4\alpha \cdot \frac{2^n (1 - \alpha)}{2^n - 1} - \frac{2^n (1 - \alpha) - 1}{2^n - 2} - \frac{2^n (1 - \alpha) - 1}{2^n - 3} + 2\alpha \cdot \frac{2^n \cdot (1 - \alpha)}{2^n - 1} - \frac{2\alpha \cdot 2^n \cdot (1 - \alpha)}{2^n - 1} \right)^2 \\
&\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 \frac{48}{2n} \alpha^2 (1 - \alpha)^2 + \frac{16}{2n} \alpha (1 - \alpha) \leq \frac{100\alpha}{2n} \tag{48}
\end{align*}
\[48\] 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 \( \alpha/100 \) of its expectation with probability 0.99. Together with equations (44) and (45) this implies the statement of the lemma.

**Theorem 6** For any sufficiently large \( n \), any \( \delta \) satisfying \( 1/n \leq \delta \leq 1/2 \) and any \( \alpha_0 \) satisfying \( 10^5/2^n \leq \alpha_0 \leq 1/1200 \), let \( \mathcal{G} \) be an algorithm that given access to a function \( f: \{0, 1\}^n \to \{0, 1\} \) with probability at least 0.99 outputs NO if it is given access to a function 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 7.1. Therefore, from Lemma 7.1 it is the case that:

\[
P_{f \in H^{10\alpha_0}/B_1} \left[ \alpha_0 \leq NS_\delta[f] \leq 30\alpha_0 \right] \geq 0.99
\]

\[
P_{f \in H^{2B_2\alpha_0}/B_2^2} \left[ 60\alpha_0 \leq NS_\delta[f] \leq 1800\alpha_0 \right] \geq 0.99
\]

Since \( \mathcal{G} \) is correct with probability at least 0.99, by a union bound it will distinguish between a random function from \( H^{10\alpha_0}/B_1 \) and \( H^{2B_2\alpha_0}/B_2^2 \) 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.

\[\blacksquare\]

**8 Appendix C**

**8.1 Proof of Lemma 2.1**

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 \( n \):

---

38
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 prove 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:

\[
1 \leq \frac{\binom{n}{n/2}}{\binom{n}{l_2}} \leq 1 + \xi \quad \quad 1 - \xi \leq \frac{\binom{n}{l_1}}{\binom{n}{n/2}} \leq 1
\]

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

### 8.2 Proof of Lemma 3.3

Recall that \( NS_{\delta}[f] = 2 \cdot Pr_{D}[f(x) = 1 \land f(z) = 0] \) and \( p_{APB} = Pr_{D}[f(x) = 1 \land f(z) = 0 \land \overline{E_1} \land \overline{E_2}] \), which implies the left inequality. We now prove the right one. We have:

\[
\frac{NS_{\delta}[f]}{2} = Pr_{D}[f(x) = 1 \land f(z) = 0] \leq Pr_{D}[f(x) = 1 \land f(z) = 0 \land \overline{E_1} \land \overline{E_2}] + Pr_{D}[E_1 \lor E_2] \tag{49}
\]

By Chernoff bound we have:

\[
Pr_{D}[E_1] \leq \exp\left(-\frac{1}{3} n \cdot 3\eta_2 \log n\right) \leq \frac{1}{n^{\eta_2}} \tag{50}
\]

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

\[
Pr_{D}[E_2]\overline{E_1} \leq Pr_{D}[|L(x) - n/2| \geq t_1 - t_2] \leq 2 \exp(-2(\eta_1 - 1)^2 \log n) = \frac{2}{n^2(\eta_1 - 1)^2} \tag{51}
\]
Thus, combining Equations (49) and (50) with Observation 2.3 we have:

\[
Pr[D(E_1 \lor E_2) \leq Pr[D[E_1] + Pr[D[E_2] \mid E_1] \leq \frac{1}{n^2} + \frac{1}{n^2(n-1)^2} \leq \frac{\epsilon}{15n^2} \tag{52}
\]

Combining Equations (49) and (52) we get:

\[
\frac{1}{2} NS_\delta[f] \leq Pr[D[f(x) = 1 \land f(z) = 0 \land E_1 \land E_2] \leq \frac{\epsilon}{15n^2}
\]

Since \( NS_\delta[f] \geq \frac{1}{n^2} \) and \( p_{AB} = Pr[D[f(x) = 1 \land f(z) = 0 \land E_1 \land E_2] \), this implies the lemma.

8.3 Proof of Lemma 3.11

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

By Chernoff bound:

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

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

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 \( O(1/(\epsilon^2 \phi)) \).

8.4 Proof of Lemma 3.12

Recall that by Equation (3):

\[
p_{AB} = \sum_{e \in E_I \cap M} p_e q_e
\]

By Lemma 3.5

\[
\left(1 - \frac{\epsilon}{310}\right) \frac{\delta}{2n} \sum_{e \in E_I \cap M} q_e \leq p_{AB} \leq \left(1 + \frac{\epsilon}{310}\right) \frac{\delta}{2n} \sum_{e \in E_I \cap M} q_e
\]

Dividing this equation by the equation in Lemma 3.9 and substituting \( |E_I| = 2^{n-1}I[f] \):

\[
\left(1 - \frac{\epsilon}{70}\right) \frac{1}{|E_I|} \sum_{e \in E_I \cap M} q_e \leq p_B \leq \left(1 + \frac{\epsilon}{70}\right) \frac{1}{|E_I|} \sum_{e \in E_I \cap M} q_e
\]
Now applying Observation 3.2:

\[
(1 - \frac{\epsilon}{33}) \frac{1}{|E_I \cap M|} \sum_{e \in E_I \cap M} q_e \leq p_B \leq \left(1 + \frac{\epsilon}{33}\right) \frac{1}{|E_I \cap M|} \sum_{e \in E_I \cap M} q_e \tag{53}
\]

Define \(\Psi\) 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 \(\Psi\), we have that \(f(x) \neq f(z)\), therefore these are the only paths contributing to \(\phi\).

Using this definition, we have:

\[
\phi = \sum_{e \in E_I \cap M} \left(Pr_{e' \in R_A}[e' = e] \sum_{(P'_1, P'_2) \in \Psi; e \in P'_1} Pr_{B,e}[(P_1 = P'_1) \land (P_2 = P'_2)]\right) \tag{54}
\]

\[
q_e = \sum_{(P'_1, P'_2) \in \Psi; e \in P'_1} Pr_{D}[(P_1 = P'_1) \land (P_2 = P'_2)]((e \in P_1)) \land \bar{E}_1 \land \bar{E}_2 \tag{55}
\]

Combining Equation (55) and Lemma 3.8 we get:

\[
\left(1 - \frac{\epsilon}{70}\right) \sum_{(P'_1, P'_2) \in \Psi; e \in P_1} Pr_{B(e)}[(P_1 = P'_1) \land (P_2 = P'_2)] \leq q_e \leq \left(1 + \frac{\epsilon}{70}\right) \sum_{(P'_1, P'_2) \in \Psi; e \in P_1} Pr_{B(e)}[(P_1 = P'_1) \land (P_2 = P'_2)] \tag{56}
\]

Now, if in Lemma 3.4 we fix an \(e_2\) and sum over \(e_1\) in \(E \cap M\) we get that for all \(e\):

\[
(1 - \frac{\epsilon}{70}) Pr_{e' \in R_A}[e' = e] \leq \frac{1}{|E \cap M|} \leq (1 + \frac{\epsilon}{70}) Pr_{e' \in R_A}[e' = e] \tag{57}
\]

Combining Equation (54) with Equation (56) and Equation (57) we get:

\[
\left(1 - \frac{\epsilon}{33}\right) \phi \leq \frac{1}{|E_I \cap M|} \sum_{e \in E_I \cap M} q_e \leq \left(1 + \frac{\epsilon}{33}\right) \phi \tag{58}
\]

Equations (53) and (58) together imply the lemma.
9 Appendix D

9.1 Proof of Lemma 4.1

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 $\frac{1}{n^{C_1}}$ fraction of its points. Such a transformation cannot increase the bias by more than $\frac{1}{n^{C_1}}$. 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 $\frac{1}{2^{n-1}}$. Therefore, the probability that $f(x) \neq f(x^{\oplus i})$ cannot change by more than $\frac{1}{2^{n-1}}$. This implies that the influence cannot change by more than $n/2^{n-1}$.

Regarding noise sensitivity, the situation is analogous. For any $\delta$, the probability that any of $x$ and $y$ equals $x_0$, on which the value of the function is changed, is at most $\frac{1}{2^{n-1}}$ 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:

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

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

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

9.2 Proof of Lemma 4.2

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

$$Pr_{x \in R_{\{0,1\}^n}}[L(x) \geq \frac{n}{2} + k\sqrt{n \log n}] \geq \frac{1}{n^{C_1}}$$

(59)

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

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

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 (59) as:

$$B[f_0] + \frac{1}{2^n} \cdot \left(\frac{n}{n/2 + k\sqrt{n \log n}}\right) > \frac{1}{n^{C_1}}$$

(60)
Additionally, for sufficiently large \( n \) we have:

\[
B[f_0] = \frac{1}{2^n} \sum_{l=n/2 + k\sqrt{n \log n} + 1}^{n} \binom{n}{l} \geq \frac{1}{2^n} \left( \frac{n}{2} + k\sqrt{n \log n} + 1 \right)
\]

\[
= \frac{1}{2^n} \left( \frac{n}{2} + k\sqrt{n \log n} \right) \frac{n - (n/2 + k\sqrt{n \log n} + 1) + 1}{n/2 + k\sqrt{n \log n} + 1} = \left( 1 - O\left( \frac{\log n}{n} \right) \right) \frac{1}{2^n} \left( \frac{n}{2} + k\sqrt{n \log n} \right) \\
\geq \frac{1}{2} \cdot \frac{1}{2^n} \left( \frac{n}{2} + k\sqrt{n \log n} \right)
\]

Above, we used the fact that \( k \) is at most a constant. Combining Equations \( 60 \) and \( 61 \) we get that for sufficiently large \( n \):

\[
3B[f_0] \geq \frac{1}{n^{c_1}} - \frac{1}{2^{n}} \left( \frac{n}{2} + k\sqrt{n \log n} \right) + 2 \cdot \frac{1}{2} \cdot \frac{1}{2^n} \left( \frac{n}{2} + k\sqrt{n \log n} \right) = \frac{1}{n^{c_1}}
\]

This together with the fact that \( B[f_0] \leq 1/n^{c_1} \), proves (b).

Consider (c) now. We have that:

\[
\frac{1}{3n^{c_1}} \leq B[f_0] = \frac{1}{2^n} \sum_{l=n/2 + k\sqrt{n \log n} + 1}^{n} \binom{n}{l} \leq \frac{n}{2^n} \left( \frac{n}{2} + k\sqrt{n \log n} + 1 \right)
\]

This implies that:

\[
Pr_{x \in \{0,1\}^n}[L(x) = n/2 + k\sqrt{n \log n} + 1] = \frac{1}{2^n} \left( \frac{n}{2} + k\sqrt{n \log n} + 1 \right) \geq \frac{1}{3n^{c_1+1}}
\]

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:

\[
I[f_0] = n \cdot Pr_{x \in \{0,1\}^n, i \in [n]}[f(x) \neq f(x^{\oplus i})] \geq n \cdot Pr_{x \in \{0,1\}^n}[L(x) = n/2 + k\sqrt{n \log n} + 1] \cdot \frac{1}{2} \\
\geq n \cdot \frac{1}{3n^{c_1+1}} \cdot \frac{1}{2} = \Omega\left( \frac{1}{n^{c_1}} \right)
\]

This proves the left inequality in (c). The right inequality is also correct, because it follows from Lemma 4.1 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 6.2) implies that noise sensitivity is an increasing function of \( \delta \). 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 \( n \cdot \frac{1}{n^2} (1 - 1/n)^{n-1} = \Omega(1) \). 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:

\[
NS_\delta[f_0] \geq NS_{1/n}[f_0] = Pr_{(x,y) \in [n]}[f_0(x) \neq f_0(y)] \\
\geq \Omega(1) \cdot Pr_{x \in \{0,1\}^n, i \in [n]}[f_0(x) \neq f_0(x^{\oplus i})] = \Omega(1) \cdot \frac{1}{n} \cdot I[f_0]
\]
Together with (c), this implies the left inequality in (d).

Regarding the right inequality, it follows from Lemma 4.1 by picking \( F \) to be the all-zeros function. Thus, (d) is true.