

# Relative-error monotonicity testing

Xi Chen <sup>\*</sup>    Anindya De <sup>†</sup>    Yizhi Huang <sup>‡</sup>    Yuhao Li <sup>§</sup>    Shivam Nadimpalli <sup>¶</sup>  
 Rocco A. Servedio <sup>||</sup>    Tianqi Yang <sup>\*\*</sup>

## Abstract

The standard model of Boolean function property testing is not well suited for testing *sparse* functions which have few satisfying assignments, since every such function is close (in the usual Hamming distance metric) to the constant-0 function. In this work we propose and investigate a new model for property testing of Boolean functions, called *relative-error testing*, which provides a natural framework for testing sparse functions.

This new model defines the distance between two functions  $f, g : \{0, 1\}^n \rightarrow \{0, 1\}$  to be

$$\text{rel-dist}(f, g) := \frac{|f^{-1}(1) \triangle g^{-1}(1)|}{|f^{-1}(1)|}.$$

This is a more demanding distance measure than the usual Hamming distance  $|f^{-1}(1) \triangle g^{-1}(1)|/2^n$  when  $|f^{-1}(1)| \ll 2^n$ ; to compensate for this, algorithms in the new model have access both to a black-box oracle for the function  $f$  being tested and to a source of independent uniform satisfying assignments of  $f$ .

In this paper we first give a few general results about the relative-error testing model; then, as our main technical contribution, we give a detailed study of algorithms and lower bounds for relative-error testing of *monotone* Boolean functions. We give upper and lower bounds which are parameterized by  $N = |f^{-1}(1)|$ , the sparsity of the function  $f$  being tested. Our results show that there are interesting differences between relative-error monotonicity testing of sparse Boolean functions, and monotonicity testing in the standard model. These results motivate further study of the testability of Boolean function properties in the relative-error model.

<sup>\*</sup>Columbia University. Email: [xichen@cs.columbia.edu](mailto:xichen@cs.columbia.edu).

<sup>†</sup>University of Pennsylvania. Email: [de.anindya@gmail.com](mailto:de.anindya@gmail.com).

<sup>‡</sup>Columbia University. Email: [yizhi@cs.columbia.edu](mailto:yizhi@cs.columbia.edu).

<sup>§</sup>Columbia University. Email: [yuhao@cs.columbia.edu](mailto:yuhao@cs.columbia.edu).

<sup>¶</sup>MIT. Email: [shivamn@mit.edu](mailto:shivamn@mit.edu).

<sup>||</sup>Columbia University. Email: [rocco@cs.columbia.edu](mailto:rocco@cs.columbia.edu).

<sup>\*\*</sup>Columbia University. Email: [tianqi@cs.columbia.edu](mailto:tianqi@cs.columbia.edu).

# 1 Introduction

Over the past several decades, property testing of Boolean functions has blossomed into a rich field with close connections to a number of important topics in theoretical computer science including sublinear algorithms, learning theory, and the analysis of Boolean functions [BLR93, Rub06, GGR98, Fis01, O'D14, Ron08]. The touchstone problems in Boolean function property testing, such as monotonicity testing and junta testing, have been intensively studied in a range of different models beyond the “standard” Boolean function property testing model which is described below; these include distribution-free testing, tolerant testing, active testing, and sample-based testing [HK07, PRR06, BBBY12, KR00].

The present work makes two main contributions: At a conceptual level, we introduce and advocate a natural new model for property testing of Boolean functions extending the standard model, which we call *relative-error property testing*. At a technical level, we give a detailed study of the problem of *monotonicity testing* in this new relative-error model.

**Motivation.** In the standard model of Boolean function property testing, a testing algorithm gets black-box access to an arbitrary unknown function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  and its goal is to tell whether  $f$  has the property or has distance at least  $\epsilon$  from every function satisfying the property, where the distance between two Boolean functions  $f, g$  is taken to be the Hamming distance

$$\text{ham-dist}(f, g) := \frac{|f^{-1}(1) \triangle g^{-1}(1)|}{2^n},$$

i.e. the fraction of points in  $\{0, 1\}^n$  on which they disagree. This model is closely analogous to the standard “dense graph” property testing model [GGR98], in which the testing algorithm may make black-box queries for entries of the unknown graph’s adjacency matrix and the distance between two undirected  $n$ -node graphs is the fraction of all  $\binom{n}{2}$  possible edges that are present in one graph but not the other.

As is well known, the standard dense graph property testing model is not well suited to testing *sparse* graphs, since under the Hamming distance measure mentioned above every sparse graph with  $o(n^2)$  edges has distance  $o(1)$  from the  $n$ -node empty graph with no edges. Consequently, alternative models were developed, with different distance measures, for testing sparse graphs; these include the bounded-degree graph model (see [GR02] and Chapter 9 in [Gol10]) and the general graph model (see [PR02] and Chapter 10 in [Gol10]).

What if we are interested in testing *sparse* Boolean functions, i.e. functions  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  that have  $|f^{-1}(1)| \ll 2^n$ ? Analogous to the situation for graphs that was described above, every sparse Boolean function is Hamming-distance-close to the constant-0 function, so the standard Boolean function property testing model is not well suited for testing sparse functions.

**Relative-error property testing.** We propose a new framework, which we call *relative-error property testing*, which is well suited for testing sparse Boolean functions. The relative-error property testing model differs from the standard model in the following ways:

- First, we define the *relative distance*<sup>1</sup> from a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  to a function  $g : \{0, 1\}^n \rightarrow \{0, 1\}$  to be

$$\text{rel-dist}(f, g) := \frac{|f^{-1}(1) \triangle g^{-1}(1)|}{|f^{-1}(1)|}, \quad \text{i.e.} \quad \text{rel-dist}(f, g) = \text{ham-dist}(f, g) \cdot \frac{2^n}{|f^{-1}(1)|}.$$

Relative-error testing uses relative distance, rather than Hamming distance, to measure distances between functions.

- Second, in the relative-error testing model, the testing algorithm has access to two different oracles: (i) a *black-box* (also called *membership query*) oracle  $\text{MQ}(f)$  as in the standard model, which is queried with a point  $x$  and returns  $f(x) \in \{0, 1\}$ ; and also (ii) a *sample oracle*  $\text{Samp}(f)$ , which takes no input and, when queried, returns a uniform random  $x \sim f^{-1}(1)$ .

**Motivation and rationale for the relative-error testing model.** The motivation for a testing model which can handle sparse functions is clear: just as there are many interesting graphs which are sparse, there are many

<sup>1</sup>See the beginning of Section 2 for a discussion of some of the basic properties of this definition.

interesting Boolean functions which have relatively few satisfying assignments. (This is especially evident when we view a Boolean function as a classification rule which outputs 1 on positive examples of some phenomenon which may be elusive; for example, generic inputs might correspond to generic candidate drugs or molecules, while positive examples correspond to candidates which have some rare but sought-after characteristic.) The distance measure  $\text{rel-dist}(f, g)$  is natural to use when studying sparse Boolean functions, since it captures the difference between  $f$  and  $g$  “at the scale” of the sparse function  $f$ , whatever that scale may be.

We observe that while the relative distance between two Boolean functions is not symmetric (i.e.  $\text{rel-dist}(f, g)$  is not necessarily equal to  $\text{rel-dist}(g, f)$ ), this does not pose a problem for us. The reason is that we are interested in the setting where  $\text{rel-dist}(f, g)$  is small, and it is easily verified that if  $\text{rel-dist}(f, g) = \varepsilon$  is at most (say) 0.99, then  $\text{rel-dist}(g, f)$  is also  $O(\varepsilon)$ ; so in the regime of interest to us, relative distance is symmetric “up to constant factors.” We further observe that in the regime of our interest, the definition of relative distance is quite robust: as long as  $\text{rel-dist}(f, g)$  is at most a small constant, then replacing  $|f^{-1}(1)|$  in the denominator by any of  $|g^{-1}(1)|$ ,  $|f^{-1}(1)| + |g^{-1}(1)|$ ,  $\max\{|f^{-1}(1)|, |g^{-1}(1)|\}$  or  $\min\{|f^{-1}(1)|, |g^{-1}(1)|\}$  only changes the value of  $\text{rel-dist}(f, g)$  by a small constant factor.

The presence of a  $\text{Samp}(f)$  oracle in the relative-distance model makes it possible to have non-trivial testing algorithms for sparse functions, since if only a black-box oracle were available then a huge number of queries could be required in order to find any input on which the unknown sparse function outputs 1. (Indeed, it is difficult to imagine a reasonable testing scenario in which the testing algorithm does not have some kind of access to positive examples of the function being tested.)

**Relation to the standard model.** Simple arguments which we give in Section 3 show that if a property is efficiently relative-error testable then it is also efficiently testable in the standard model, and moreover that there are properties which are efficiently testable in the standard model but not in the relative-error model. Thus, relative-error testing is at least as hard as standard-model testing. A wide range of natural questions about the testability of well-studied Boolean function properties present themselves for this model; we discuss some of these questions at the end of this introduction. The main technical results of this paper, though, deal with relative-error testing of *monotonicity*, which is one of the most thoroughly studied Boolean function properties [GGL<sup>+</sup>00, FLN<sup>+</sup>02, CS13a, CS13b, CST14, CDST15, KMS18]. (Recall that a Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  is *monotone* if whenever  $x, y \in \{0, 1\}^n$  have  $x_i \leq y_i$  for all  $i$ , i.e.  $x \preceq y$ , it holds that  $f(x) \leq f(y)$ .)

**1.1 Our results Comparison with standard-error model:** We start in Section 3 by noting some basic points of comparison between the relative-error property testing model vis-a-vis the standard-error property testing model for general properties of Boolean functions. First of all, it is not too hard to establish (Fact 3.1) that any property  $\mathcal{C}$  which is testable in the relative-error model is also testable in the standard-error model. In the other direction, any algorithm to test a property  $\mathcal{C}$  in the standard error model can be used to test  $\mathcal{C}$  in the relative error model (see Fact 3.2). However, the overhead of this simulation grows as  $1/p$ , where  $p := |f^{-1}(1)|/2^n$  is the density of the target function  $f$ . Thus the most interesting and challenging regime for relative error property testing is when the target function is sparse — i.e.,  $p$  is vanishing as a function of  $n$  ( $p$  may even be as small as  $2^{-\Theta(n)}$ ).

**Upper and lower bounds for monotonicity testing:** We focus on monotonicity testing in the relative-error model. Using the analysis of the “edge tester” [GGL<sup>+</sup>00] for monotonicity testing in the standard model, it is easy to obtain a one-sided<sup>2</sup>, non-adaptive<sup>3</sup> algorithm with  $O(n/\varepsilon)$  queries in the relative-error model (see Section 4.1 as a warmup). Our main algorithm as stated below, on the other hand, makes  $O(\log(N)/\varepsilon)$  queries, where  $N := |f^{-1}(1)|$  denotes the *sparsity* of  $f$  and is *not* given to the algorithm. Note that one always has  $\log(N)/\varepsilon \leq n/\varepsilon$  as  $N \leq 2^n$  trivially but the former can be asymptotically lower when  $f$  is sparse (e.g.,  $n^{1/3}/\varepsilon$  when  $N = 2^{n^{1/3}}$ ).

<sup>2</sup>Recall that a one-sided tester for a class of functions is one which must accept (with probability 1) any function in the class. This is in contrast to making two-sided error, where an algorithm may reject a function in the class with small probability.

<sup>3</sup>A non-adaptive algorithm is one in which the choice of its  $i$ -th query point does not depend on the responses received to queries  $1, \dots, i-1$ .

**THEOREM 1.1. (TESTING ALGORITHM)** *There is a one-sided adaptive algorithm which is an  $\varepsilon$ -relative-error tester for monotonicity (i.e., it always returns “monotone” when the input function  $f$  is monotone and returns “not monotone” with probability at least  $2/3$  when  $f$  has relative distance at least  $\varepsilon$  from monotone); with probability at least  $1 - \delta$ , it makes no more than  $O(\log(1/\delta)/\varepsilon + \log(N)/\varepsilon)$  calls to  $\text{Samp}(f)$  and  $\text{MQ}(f)$ .*

We remark that while the algorithm described in Theorem 1.1 is adaptive, it can be made non-adaptive and still makes  $O(\log(N)/\varepsilon)$  queries (in the worst case) when an estimate of  $N$  is given as part of the input to the algorithm. Our first main lower bound shows that this is indeed tight:  $\tilde{\Omega}(\log N)$  queries are needed for non-adaptive algorithms even when an estimate of  $N$  is given.

**THEOREM 1.2. (NON-ADAPTIVE LOWER BOUND)** *For any constant  $\alpha_0 < 1$ , there exists a constant  $\varepsilon_0 > 0$  such that any two-sided, non-adaptive algorithm for testing whether a function  $f$  with  $|f^{-1}(1)| = \Theta(N)$  for some given parameter  $N \leq 2^{\alpha_0 n}$  is monotone or has relative distance at least  $\varepsilon_0$  from monotone must make  $\Omega(\log N)$  queries.*

Finally, we show that  $\tilde{\Omega}((\log N)^{2/3})$  queries are needed for adaptive algorithms:

**THEOREM 1.3. (ADAPTIVE LOWER BOUND)** *For any constant  $\alpha_0 < 1$ , there exists a constant  $\varepsilon_0 > 0$  such that any two-sided, adaptive algorithm for testing whether a function  $f$  with  $|f^{-1}(1)| = \Theta(N)$  for some given parameter  $N \leq 2^{\alpha_0 n}$  is monotone or has relative distance at least  $\varepsilon_0$  from monotone must make  $\Omega((\log N)^{2/3})$  queries.*

Theorem 1.2 and Theorem 1.3 also imply that, as functions of  $n$ ,  $\tilde{\Omega}(n)$  and  $\tilde{\Omega}(n^{2/3})$  queries are needed for non-adaptive and adaptive relative-error monotonicity testing algorithms, respectively.

Summarizing our results, we see that there is both a correspondence and a significant point of difference between the state-of-the-art for standard-model monotonicity testing upper and lower bounds [CWX17, KMS18] and the upper and lower bounds for relative-error monotonicity testing that we establish. (We fix the error parameter  $\varepsilon$  to be a constant, for simplicity, in the discussion below.) In both cases there is a  $3/2$ -factor-gap between the exponent of the best known adaptive lower bound and the best known algorithm, but the source of this gap is intriguingly different between the two cases. In the standard model, the best known algorithm is the sophisticated “path tester” of Khot et al. [KMS18] which makes  $\tilde{O}(\sqrt{n})$  queries, while the strongest lower bound (for general testers, i.e. adaptive testers with two-sided error) is  $\tilde{\Omega}(n^{1/3})$ , due to Chen et al. [CWX17]. In contrast, in the relative-error setting the best algorithm we know of is a variant of the simple “edge tester” (given in Section 4), which achieves an  $O(\log(N))$  complexity. But in the relative-error setting, for a wide range of values of  $N = |f^{-1}(1)|$ , it is possible to prove a *stronger*  $\tilde{\Omega}((\log N)^{2/3})$  lower bound than in the standard model. So the gap in the exponent is again a factor of  $3/2$ ; but in the relative-error setting the state-of-the-art algorithm is a simple one (in contrast with the sophisticated algorithm and analysis, based on isoperimetry, from [KMS18]), and a stronger lower bound is achievable in the relative-error model than in the standard model.

**1.2 Technical Overview** We now discuss techniques underlying our upper and lower bounds for relative-error monotonicity testing, with the goal of giving some intuition for how the bounds are proved.

**Upper bounds.** We begin with the relative-error monotonicity testing algorithms. Our algorithm is based on the “edge tester” for the standard monotonicity testing problem; the high-level idea is to look for an explicit *edge* of the Boolean hypercube that witnesses a violation of monotonicity.

We begin by recalling the standard edge tester of [GGL<sup>+</sup>00]. It simply repeats the following test  $O(n/\varepsilon)$  times: sample an  $\mathbf{x}$  uniformly at random from  $\{0, 1\}^n$ , sample a  $\mathbf{y}$  “immediately above”  $\mathbf{x}$  in the Boolean hypercube<sup>4</sup>, and reject if  $f(\mathbf{x}) > f(\mathbf{y})$ . Note that such a rejection only happens if  $(\mathbf{x}, \mathbf{y})$  violates monotonicity, so the algorithm will never reject a monotone function  $f$ . To show that the algorithm will reject any  $f$  that is far from monotone with high probability, [GGL<sup>+</sup>00] showed that if at least  $\varepsilon 2^n$  points need to be changed to make  $f$  monotone, then  $f$  must have at least  $\varepsilon 2^{n-1}$  such violating edges. Since there are  $n 2^{n-1}$  edges in total in the Boolean hypercube, the probability that the edge tester hits such a violating edge is at least  $\varepsilon/n$ .

We first describe a relative-error testing algorithm that is assumed to know the sparsity  $N$  (i.e. the sparsity  $N$  is given to it as an input). This algorithm first samples an  $\mathbf{x} \in f^{-1}(1)$  using the sampling oracle  $\text{Samp}(f)$ , then samples a point  $\mathbf{y}$  that is immediately above  $\mathbf{x}$ , and rejects if  $f(\mathbf{y}) = 0$ . One can see that the only difference between this algorithm and the classical edge tester is that instead of sampling  $\mathbf{x}$  uniformly from all of  $\{0, 1\}^n$ , we

<sup>4</sup>Formally,  $\mathbf{y}$  is a point with  $\mathbf{x} \prec \mathbf{y}$  and  $\|\mathbf{y}\|_1 = \|\mathbf{x}\|_1 + 1$ , where  $\|\mathbf{x}\|_1$  denotes the Hamming weight of  $\mathbf{x} \in \{0, 1\}^n$ .

sample  $x$  uniformly from  $f^{-1}(1)$ . Our analysis shows that the algorithm will find a violation with probability at least  $\Omega(\varepsilon/(\log N))$  when  $f$  is  $\varepsilon$ -relative-error-far from monotone.

The key observation for the analysis is the following: if a monotone function  $f$  has sparsity  $N$ , then any point  $x \in f^{-1}(1)$  must have  $\|x\|_1 \geq n - \log N$ , since every point above  $x$  must also be in  $f^{-1}(1)$ . For each  $x$  having  $\|x\|_1 \geq n - \log N$ , there will be at most  $\log N$  many possible  $y$ 's "immediately above"  $x$ , and hence our relative-error testing algorithm is sampling from at most  $N \log N$  possible  $(x, y)$  pairs. Applying the results of [GGL<sup>+</sup>00], we know that when  $f$  is  $\varepsilon$ -relative-error-far from monotone,  $f$  has at least  $\varepsilon N/2$  many violating  $(x, y)$  edges. Therefore, the probability that our algorithm will find a violation is at least  $\varepsilon/(2 \log N)$ .

Now we consider the more challenging situation that the algorithm does not know the sparsity  $N$ . Our approach is to first obtain (an estimate of) the minimum Hamming weight of all points in  $f^{-1}(1)$  by drawing a few samples from the sampling oracle, and then run the algorithm sketched above. In more detail, suppose that  $x$  is the sampled point with minimum Hamming weight, which is  $n - k$ ; our algorithm samples a few uniform random points from all points that are above  $x$ . If any of them have  $f = 0$  then the algorithm rejects, and otherwise it uses  $k$  as a proxy for  $\log N$  in the algorithm sketched above. To see why this works, it suffices to consider the case that  $f$  is  $\varepsilon$ -relative-error-far from monotone. Observe first that if any sampled  $x \in f^{-1}(1)$  has weight  $\|x\|_1 < n - 2 \log N$ , then most points above  $x$  cannot be in  $f^{-1}(1)$  (since there are  $N^2$  points above  $x$  and only  $N$  satisfying assignments in total), so sampling a few uniform random points above  $x$  will reveal a violation of monotonicity. So we may assume that the sampled point  $x$  with minimum Hamming weight has Hamming weight at least  $n - 2 \log N$ . Now our earlier analysis shows that the earlier algorithm will indeed reject any  $f$  that has relative error at least  $\varepsilon$  for monotonicity.

**Lower bounds.** We consider the case when the relative distance parameter  $\varepsilon$  is a constant. The upper bounds show that even when  $N$  is much smaller than  $2^n$ , there are algorithms for relative error testing that make only  $O(\log N)$  queries. We describe some of the key new ideas behind our constructions for the  $\tilde{\Omega}(\log N)$  lower bound for non-adaptive algorithms (Theorem 1.2, Section 6) and the  $\tilde{\Omega}((\log N)^{2/3})$  lower bound for adaptive algorithms (Theorem 1.3, Section 7).

Intuitively, to make monotonicity testing as hard as possible, we would like to use functions that have as little poset structure of the cube as possible. (For example, all monotonicity lower bounds in the standard model use functions that are only nontrivial in the *middle layers*, i.e., points with Hamming weight  $(n/2) \pm O(\sqrt{n})$ , with all points above the middle layers set to 1 and all points below set to 0 by default. This way only the poset structure in the middle  $O(\sqrt{n})$  layers is relevant.) Towards this end, our lower bound constructions use functions which are only nontrivial on two adjacent layers of the Boolean hypercube, we call them *two-layer* functions (see Section 5). More precisely, a function  $f$  is a two-layer function if it is only nontrivial on points in layers  $3n/4$  and  $3n/4 + 1$ : every point with weight  $< 3n/4$  is set to 0 and every point with weight  $> 3n/4 + 1$  is set to 1. (We use the constant  $3/4$  just to make the presentation more concrete; it can be replaced by any constant strictly between  $1/2$  and  $1$ .)

In addition to the much simplified poset structure (which we discuss in the paragraph about our construction below), two advantages of using two-layer functions follow from simple calculations: (1) For  $f$  to have  $\varepsilon$ -relative error from monotonicity for some constant  $\varepsilon$ , it suffices to show that a constant fraction of points in the two layers  $3n/4$  and  $3n/4 + 1$  must be changed to make the function monotone; (2) The use of two-layer functions let us reduce the analysis of general relative-error testing algorithms (which can use both MQ and Samp oracles) to algorithms that use MQ only (see Claim 5.2).

So the main technical challenge is to establish strong lower bounds for two-layer functions. Recall that for monotonicity testing in the standard model, state of the art lower bounds are obtained using the Talagrand DNF [BB16, CWX17] (a random  $2^{\sqrt{n}}$ -term DNF in which each term has  $\sqrt{n}$  variables sampled uniformly at random) and a depth-3 extension of the Talagrand DNF [CWX17]. The  $\sqrt{n}$  (in both the size of each term and the exponent of the number of terms) comes from the fact we mentioned early about the middle  $O(\sqrt{n})$  layers: if  $\sqrt{n}$  were replaced by a larger polynomial  $n^{0.5+c}$ , then most layers in the middle would become trivial (points in layers above  $(n/2) + O(n^{0.5-c})$  would be most likely set to 1, and points in layers below  $(n/2) - O(n^{0.5-c})$  would be most likely set to 0).

In contrast, since we only focus on two-layer functions, the obstacle described above is no longer there, and it turns out that we can use a novel variant of the Talagrand DNF with  $2^{\Theta(n)}$  many terms each of size  $\Theta(n)$ . This is the construction that leads to our non-adaptive  $\tilde{\Omega}(\log N)$  lower bound for relative-error monotonicity testing. For the  $\tilde{\Omega}((\log N)^{2/3})$  adaptive lower bound, our construction uses a corresponding variant of the depth-3 extension of Talagrand DNF. Using two-layer functions also necessitates various other technical modifications of

the [BB16, CWX17] constructions, which are detailed in Section 6 and Section 7.

**1.3 Related work** Several earlier works have considered property testing models which are similar to our model of relative-error testing of Boolean functions  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ . One such work is an early paper of Rademacher and Vempala [RV04], who considered essentially a continuous-domain analogue of our framework. [RV04] studied the problem of testing whether an unknown set  $S \subset \mathbb{R}^n$  is convex versus  $\varepsilon$ -far from convex, where a finite-volume set  $S \subseteq \mathbb{R}^n$  is said to be  $\varepsilon$ -far from convex if  $\text{Leb}(S \triangle C) \geq \varepsilon \cdot \text{Leb}(S)$  for every convex set  $C$ , where  $\text{Leb}(\cdot)$  denotes the Lebesgue volume. The query model considered in that work, like our query model, is that the algorithm is given access both to a black-box oracle  $\text{MQ}(S)$  for (the indicator function of) the unknown set  $S$ , as well as access to a  $\text{Samp}(S)$  oracle which when queried outputs a uniform random element of  $S$ .

[RV04] gave an  $(n/\varepsilon)^{O(n)}$ -complexity algorithm for testing convexity in their model, and also gave an exponential lower bound for a specific convexity tester known as the “line segment tester”; the [RV04] lower bound was strengthened and extended to an exponential lower bound for a “convex hull tester” in recent work of Blais and Bommireddi [BB20]. Neither of the works [RV04, BB20] studied general algorithms for relative-error property testing or considered relative-error property testing of functions over the Boolean domain  $\{0, 1\}^n$ , which is the focus of the present work.

A similar model, in which the relative-distance measure is employed and the testing algorithm has both  $\text{MQ}(f)$  and  $\text{Samp}(f)$  oracles, was considered by Ron and Tsur [RT14], who considered the problem of testing *sparse images* for properties such as sparsity, convexity, monotonicity, and being a line. In their setting, the domain of interest was the two-dimensional discrete grid  $[n] \times [n]$ , rather than the high-dimensional Boolean hypercube  $\{0, 1\}^n$  that we consider, and hence the results and techniques are very different between their setting and ours.

We remark that several early works [Mag00, KMS03] studied self-testing of *real-valued* functions with a relative-error criterion, in the sense that error terms are allowed to be proportional to the particular function value being computed. This line of work, with its focus on real-valued functions, has a rather different flavor from our relative-error model of testing (sparse) Boolean functions.

**1.4 Future work** Apart from the general observations about relative-error testing given in Section 3, this work focuses on monotonicity testing. We hope that the relative-error model may open up a new perspective on Boolean function property testing more broadly, though, by giving a clean theoretical framework for studying testing of *sparse* Boolean functions for all sorts of properties. Beyond monotonicity, there are many interesting and natural questions about relative-error testing of other specific well-studied Boolean function properties; a few of the questions which seem most interesting to us are listed below.

- **Juntas:** As we will discuss after Fact 3.2, there is a relative-error testing algorithm for the class of  $k$ -juntas that makes  $O(k2^k/\varepsilon)$  black-box queries and does not use the sample oracle. This dependence on  $k$  is exponentially worse than the state-of-the-art  $O(k/\varepsilon + k \log k)$ -query  $k$ -junta testing algorithm [Bla09] for the standard model, so it is natural to ask: can the class of  $k$ -juntas be relative-error tested using  $\text{poly}(k, 1/\varepsilon)$  queries? We note that in the distribution-free model juntas are testable with  $\text{poly}(k, 1/\varepsilon)$  queries [CLS<sup>+</sup>18, Bsh19], but it is not clear how a distribution-free testing algorithm can help with relative-error testing.
- **LTFs:** As we will mention in Section 3, the class of LTFs can be  $\varepsilon$ -relative-error tested using  $\text{poly}(n, 1/\varepsilon)$  samples and queries (by a reduction to the  $\text{poly}(n, 1/\varepsilon)$ -sample relative-error learning algorithm of [DDS15]). However, in the standard property testing model it is known that the class of all linear threshold functions (LTFs) over  $\{0, 1\}^n$  can be  $\varepsilon$ -tested using only  $\text{poly}(1/\varepsilon)$  queries, independent of the ambient dimension  $n$  [MORS10]. Are LTFs relative-error testable using  $\text{poly}(1/\varepsilon)$  queries and samples, or can an  $\omega_n(1)$  lower bound be shown?
- **DNFs:** A similar question can be asked for DNFs. As we will mention in Section 3, the class of  $s$ -term DNFs can be  $\varepsilon$ -relative-error tested using  $\text{poly}(n^{\log(s/\varepsilon)})$  samples and queries (by a reduction to the  $\text{poly}(n^{\log(s/\varepsilon)})$ -sample relative-error learning algorithm of [DDS15]). However, in the standard property testing model it is known that the class of all  $s$ -term DNFs over  $\{0, 1\}^n$  can be  $\varepsilon$ -tested using only  $\text{poly}(s/\varepsilon)$  queries, independent of  $n$  [DLM<sup>+</sup>07, CGM11, Bsh20]. Are  $s$ -term DNFs relative-error testable using  $\text{poly}(s/\varepsilon)$  queries and samples, or can an  $\omega_n(1)$  lower bound (or even an  $n^{\omega_s(1)}$  lower bound) be shown?

## 2 Preliminaries

We start by defining the distance metric that we will use:

DEFINITION 1. Given two functions  $f, g : \{0, 1\}^n \rightarrow \{0, 1\}$ , the relative distance from  $f$  to  $g$  is defined as

$$\text{rel-dist}(f, g) := \frac{|f^{-1}(1) \triangle g^{-1}(1)|}{|f^{-1}(1)|}.$$

For a class  $\mathcal{C}$  of functions from  $\{0, 1\}^n$  to  $\{0, 1\}$ , the relative distance from  $f$  to  $\mathcal{C}$  is given by

$$\text{rel-dist}(f, \mathcal{C}) := \min_{g \in \mathcal{C}} \text{rel-dist}(f, g).$$

Note that the relative distance between two Boolean functions is not symmetric (i.e.  $\text{rel-dist}(f, g)$  is not necessarily equal to  $\text{rel-dist}(g, f)$ ), this does not pose a problem for us: we will chiefly be interested in cases where  $\text{rel-dist}(f, g)$  is small, and it is easily verified that if  $\text{rel-dist}(f, g) = \varepsilon$  is at most (say) 0.99, then  $\text{rel-dist}(g, f) = O(\varepsilon)$ . We further mention that in the regime of our interest, the definition of relative distance is fairly robust: as long as  $\text{rel-dist}(f, g)$  is at most a small constant, then replacing  $|f^{-1}(1)|$  in the denominator by any of  $|g^{-1}(1)|$ ,  $|f^1(1)| + |g^{-1}(1)|$ , or  $\max\{|f^{-1}(1)|, |g^{-1}(1)|\}$ , only changes the value of  $\text{rel-dist}(f, g)$  by a constant factor.

Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  be the unknown Boolean function that is being tested. Recall that a *sampling oracle* for  $f$  is an oracle which takes no inputs and, each time it is invoked, independently returns a uniform random element of  $f^{-1}(1)$ .<sup>5</sup> A *black-box oracle* for  $f$  takes as input an  $n$ -bit string  $x$  and returns  $f(x)$ ; we sometimes refer to a black-box oracle simply “oracle for  $f$ .”

DEFINITION 2. (RELATIVE-ERROR PROPERTY TESTING) Let  $\mathcal{C}$  be a class of functions from  $\{0, 1\}^n$  to  $\{0, 1\}$ . A  $q$ -query  $\varepsilon$ -relative-error property testing algorithm for  $\mathcal{C}$  is a randomized algorithm  $A$  which is given access to a sampling oracle for  $f$  and a black-box oracle for  $f$ , where  $f$  may be any (unknown) function from  $\{0, 1\}^n$  to  $\{0, 1\}$ .  $A$  makes at most  $q$  calls to the sampling oracle and at most  $q$  calls to the black-box oracle, and has the following performance guarantee:

- If  $f \in \mathcal{C}$  then with probability at least  $2/3$  algorithm  $A$  outputs “accept”;
- If  $\text{rel-dist}(f, \mathcal{C}) > \varepsilon$  then with probability at least  $2/3$  algorithm  $A$  outputs “reject.”

A one-sided relative error property testing algorithm is one which outputs “reject” with probability 0 unless  $f \notin \mathcal{C}$  (i.e. if  $f \in \mathcal{C}$  then it outputs “accept” with probability 1).

REMARK 3. We note that without loss of generality a relative-error testing algorithm can be assumed to make all  $q$  of its calls to the sampling oracle before making any calls to the black-box oracle.

REMARK 4. Our main focus of interest in this work will be testing the class  $\mathcal{C}$  of all monotone Boolean functions, which we denote  $\mathcal{C}_{\text{monotone}}$ . It is easy to verify that any  $2^{o(n)}$ -query relative-error testing algorithm for this class must use both the sampling oracle and the black-box oracle (we give a simple argument establishing this in Appendix B). Thus our framework, which allows both the sampling oracle and the black-box oracle, is a “minimal adequate model” for relative-error monotonicity testing.

## 3 Some general results on relative-error testing

**Relative-error testing implies standard-model testing.** We begin with the following straightforward fact, which shows that any property that is efficiently relative-error testable is also efficiently testable in the standard model:

FACT 3.1. Let  $\mathcal{C}$  be any class of functions from  $\{0, 1\}^n \rightarrow \{0, 1\}$ . Suppose that there is a relative-error  $\varepsilon$ -testing algorithm  $T$  for  $\mathcal{C}$  that makes  $q(\varepsilon, n)$  many black-box queries and calls to Samp. Then there is a standard-model  $\varepsilon$ -testing algorithm  $T'$  for  $\mathcal{C}$  which makes at most  $O(1/\varepsilon^2 + \frac{1}{\varepsilon}q(\varepsilon, n))$  calls to the black-box oracle.

<sup>5</sup>If  $f$  is the constant-0 function then the sampling oracle is assumed to return a special  $\perp$  symbol; note that if this happens it completely identifies the function  $f$  as the constant-0 function. Hence in our subsequent discussion we implicitly assume that the function  $f$  being tested is not the constant-0 function.

*Proof.* Let  $\alpha \in [0, 1]$  be the value of  $\text{ham-dist}(0, \mathcal{C})$ , i.e. the Hamming distance between the constant-0 function and the nearest function in  $\mathcal{C}$ ; the algorithm and its analysis will make use of this quantity. (Note that no queries to the unknown function  $f$  are required for the algorithm to determine the value of  $\alpha$ .)

The algorithm  $T'$  works as follows:

1. It first makes  $m_1 = O(1/\varepsilon^2)$  calls to the black-box oracle for  $f$  on uniform random inputs; let  $\mathbf{m}'_1 \leq m_1$  be the number of those queries that have  $f(x) = 1$ . If  $\mathbf{m}'_1/m_1 \leq \varepsilon/3$ , then  $T'$  outputs “accept” if  $\alpha \leq \varepsilon/2$  and outputs “reject” otherwise. If  $\mathbf{m}'_1/m_1 > \varepsilon/3$ , then
2.  $T'$  draws  $m_2 = \frac{C}{\varepsilon}q(\varepsilon, n)$  independent uniform random points from  $\{0, 1\}^n$  (for a suitable absolute constant  $C$ ) and queries  $f$  on each of them. If fewer than  $q(\varepsilon, n)$  of the  $m_2$  points are satisfying assignments of  $f$  then  $T'$  halts and fails. Otherwise,
3.  $T'$  uses the first  $q(\varepsilon, n)$  of the satisfying assignments as the  $\text{Samp}(f)$  responses that algorithm  $T(\varepsilon, n)$  requires. (Recall that by Definition 3, we may suppose that  $T$  makes all its calls to  $\text{Samp}(f)$  before making any calls to  $\text{MQ}(f)$ .)  $T'$  continues to simulate  $T(\varepsilon, n)$  (making at most  $q(\varepsilon, n)$  calls to  $\text{MQ}(f)$ , the same way  $T$  does) and returns what  $T'$  outputs.

The query complexity of  $T'$  is clearly as claimed. To establish correctness, we begin by observing that by a simple Chernoff bound, with probability 99/100 the value of  $\mathbf{m}'_1/m_1$  is within  $\pm\varepsilon/100$  of the true value of  $|f^{-1}(1)|/2^n$  (we will use this repeatedly in the following arguments).

**Case 1:**  $f \in \mathcal{C}$ . Suppose first that  $|f^{-1}(1)|/2^n \leq \varepsilon/4$ . In this case, with probability at least 99/100 we have  $\mathbf{m}'_1/m_1 \leq \varepsilon/4 + \varepsilon/100 = 26\varepsilon/100$ . Since  $\alpha \leq \text{ham-dist}(f, 0) = |f^{-1}(1)|/2^n \leq \varepsilon/4$ , in this case  $T'$  correctly outputs “accept” in Step 1.

So suppose next that  $f \in \mathcal{C}$  and  $\varepsilon/4 < |f^{-1}(1)|/2^n \leq \varepsilon/2$ . In this case we have  $\alpha \leq \text{ham-dist}(f, 0) \leq \varepsilon/2$  so  $T'$  does not output “reject” in Step 1. Even if  $T'$  does not output “accept” in Step 1, since  $|f^{-1}(1)|/2^n \geq \varepsilon/4$ , for a suitable choice of the constant  $C$  we have that with probability at least 99/100 the algorithm  $T'$  does not fail in Step 2. So the relative-error  $\varepsilon$ -testing algorithm  $T(\varepsilon, n)$  is executed in Step 3 and we invoke its guarantee, which is that it outputs “accept” with probability at least 2/3 (since  $f \in \mathcal{C}$ ). So in this case as well  $T'$  correctly outputs “accept” with high probability.

The remaining subcase of Case 1 is that  $f \in \mathcal{C}$  and  $|f^{-1}(1)|/2^n > \varepsilon/2$ . In this case the probability that  $T'$  outputs “reject” in Step 1 is at most 1/100 (since this only happens if  $\mathbf{m}'_1/m_1 \leq \varepsilon/3$ ), so we may suppose that  $T'$  reaches Step 2. In this case since  $|f^{-1}(1)|/2^n \geq \varepsilon/2$ , for a suitable choice of the constant  $C$  we have that with probability at least 99/100 the algorithm  $T'$  does not fail in Step 2. So as in the previous paragraph, the relative-error  $\varepsilon$ -testing algorithm  $T(\varepsilon, n)$  is executed in Step 3, and in this case as well  $T'$  correctly outputs “accept” with high probability (which can be amplified to any constant probability using standard techniques).

**Case 2:**  $\text{ham-dist}(f, \mathcal{C}) > \varepsilon$ . Suppose first that  $|f^{-1}(1)|/2^n \leq E = \varepsilon/4$ . Then in Step 1, with probability 99/100 we have  $\mathbf{m}'_1/m_1 \leq \varepsilon/4 + \varepsilon/100 = 26\varepsilon/100$ . It cannot be the case that  $\alpha \leq \varepsilon/2$  (because if  $\alpha \leq \varepsilon/2$  then we would have  $\text{ham-dist}(f, \mathcal{C}) \leq \text{ham-dist}(f, 0) + \text{ham-dist}(0, \mathcal{C}) \leq 3\varepsilon/4$ , but in Case 2 we have  $\text{ham-dist}(f, \mathcal{C}) > \varepsilon$ ), so algorithm  $T'_1$  correctly outputs “reject” with high probability in Step 1.

Suppose next that  $\text{ham-dist}(f, \mathcal{C}) > \varepsilon$  and  $\varepsilon/4 < |f^{-1}(1)|/2^n \leq \varepsilon/2$ . If  $\mathbf{m}'_1/m_1 \leq \varepsilon/3$  then similar to the previous paragraph we cannot have  $\alpha \leq \varepsilon/2$  (because if  $\alpha \leq \varepsilon/2$  then we would have  $\text{ham-dist}(f, \mathcal{C}) \leq \text{ham-dist}(f, 0) + \text{ham-dist}(0, \mathcal{C}) \leq \varepsilon$ , but in Case 2 we have  $\text{ham-dist}(f, \mathcal{C}) > \varepsilon$ ), so  $T'_1$  correctly outputs “reject” in Step 1. If  $\mathbf{m}'_1/m_1 > \varepsilon/3$  then since  $|f^{-1}(1)|/2^n \geq \varepsilon/4$ , for a suitable choice of the constant  $C$  we have that with probability at least 99/100 the algorithm  $T'$  does not fail in Step 2. So the relative-error  $\varepsilon$ -testing algorithm  $T(\varepsilon, n)$  is executed in Step 3 and we invoke its guarantee, which is that it correctly outputs “reject” with probability at least 2/3 (since  $\text{rel-dist}(f, \mathcal{C}) = \text{ham-dist}(f, \mathcal{C}) \cdot \frac{2^n}{|f^{-1}(1)|} \geq \text{ham-dist}(f, \mathcal{C}) > \varepsilon$ ). So in this case as well  $T'$  correctly outputs “reject” with high probability.

The final case when  $\text{ham-dist}(f, \mathcal{C}) > \varepsilon$  is that  $|f^{-1}(1)|/2^n > \varepsilon/2$ . In this case the probability that  $T'$  outputs “reject” in Step 1 is at most 1/100 (since this only happens if  $\mathbf{m}'_1/m_1 \leq \varepsilon/3$ ), so we may suppose that  $T'$  reaches Step 2. In this case the arguments of the previous paragraph again give us that  $T'$  correctly outputs “reject” with high probability.  $\square$

We remark that a simple example, given in Appendix A, shows that there are properties which are trivially testable in the standard model but very hard to test in the relative-error model. Together with Fact 3.1, this shows that relative-error testing is a more demanding model than the standard property testing model.



**Standard-model testing implies relative-error testing, for not-too-sparse properties.** The next result, together with Fact 3.1, implies that standard-model testing and relative-error testing are essentially equivalent for classes of functions that are “not too sparse:”

**FACT 3.2.** *Let  $p = p(n) > 0$  and let  $\mathcal{C}$  be any class of functions from  $\{0, 1\}^n \rightarrow \{0, 1\}$  such that every  $f \in \mathcal{C}$  has  $|f^{-1}(1)|/2^n \geq p$ . Suppose that there is a standard-model  $\varepsilon$ -testing algorithm  $T$  for  $\mathcal{C}$  that makes  $q(\varepsilon, n)$  many black-box queries. Then there is a relative-error  $\varepsilon$ -testing algorithm  $T'$  for  $\mathcal{C}$  which, when run on any function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , makes no calls to the sampling oracle and at most  $O(1/p + q(p\varepsilon/2, n))$  calls to the black-box oracle.*

*Proof.* Algorithm  $T'$  first makes  $m = O(1/p)$  calls to the black-box oracle for  $f$  on uniform random inputs, and rejects if fewer than  $3pm/4$  of the queried points are satisfying assignments of  $f$ . If  $T'$  does not reject in this first phase, it then performs  $O(1)$  many runs of the standard-model testing algorithm  $T$ , each time with closeness parameter  $p\varepsilon/2$ , and outputs the majority of those runs.

It is clear that the query complexity is as claimed. To establish correctness, first consider the case that  $f \in \mathcal{C}$ . Since  $|f^{-1}(1)|/2^n \geq p$ , the probability that the algorithm rejects in the first phase is at most (say)  $1/6$  by a standard multiplicative Chernoff bound. Since  $f \in \mathcal{C}$ , the probability that any individual run of  $T$  outputs “reject” is at most  $1/3$ , so the probability that the majority of the  $O(1)$  runs output “reject” is at most (say)  $1/6$ . Hence the probability that  $f$  is rejected is at most  $1/3$  as desired.

Next, suppose that  $\text{rel-dist}(f, \mathcal{C}) > \varepsilon$ . One possibility is that  $|f^{-1}(1)|/2^n \leq p/2$ ; if this is the case, then the probability that  $T$  rejects in the first phase is at least  $2/3$  as required. So consider the other possibility, which is that  $|f^{-1}(1)|/2^n > p/2$ . We have

$$\varepsilon < \text{rel-dist}(f, \mathcal{C}) = \min_{g \in \mathcal{C}} \frac{|f^{-1}(1) \triangle g^{-1}(1)|}{|f^{-1}(1)|} < \frac{2}{p} \cdot \min_{g \in \mathcal{C}} \frac{|f^{-1}(1) \triangle g^{-1}(1)|}{2^n},$$

which rearranges to give  $\text{ham-dist}(f, \mathcal{C}) \geq p\varepsilon/2$ . So even if  $T'$  makes it to the second stage, the probability that the majority of  $O(1)$  calls to  $T$  with closeness parameter  $p\varepsilon/2$  output “accept” is at most  $1/3$ .  $\square$

Fact 3.2 sheds light on relative-error testing of some classes that have been intensively studied in the standard model. For instance, since every non-constant parity function  $f$  has  $|f^{-1}(1)|/2^n = 1/2$ , the standard-model  $O(1/\varepsilon)$ -query testing algorithm for linear (parity) functions [BLR93] yields a relative-error tester with the same  $O(1/\varepsilon)$ -query complexity. As another application, since every non-constant  $k$ -junta has  $|f^{-1}(1)|/2^n \geq 1/2^k$ , the  $O(k/\varepsilon + k \log k)$ -query algorithm of Blais [Bla09] for testing juntas yields a relative-error tester for  $k$ -juntas that makes  $O(k2^k/\varepsilon)$  black-box queries. On the other hand, Fact 3.2 does not give anything for some other natural classes such as LTFs,  $s$ -term DNF formulas, and monotone functions.

**Relative-error learning implies relative-error testing.** Several recent works [DDS15, CDS20] have studied the problem of learning the distribution of satisfying assignments of an unknown Boolean function. In this framework, a learning algorithm for a concept class  $\mathcal{C}$  of Boolean functions over  $\{0, 1\}^n$  has access to uniform random satisfying assignments (i.e. to a sample oracle for the unknown target function  $f \in \mathcal{C}$ ), and the goal of the learner is to output an  $\varepsilon$ -sampler for  $f^{-1}(1)$ , which is a circuit that, when given independent uniform random bits as its input, outputs a draw from a distribution  $\mathcal{D}$  that has total variation distance at most  $\varepsilon$  from the uniform distribution over  $f^{-1}(1)$ . Known algorithms in this framework work in two stages:

1. First, they perform  $\varepsilon/2$ -relative-error proper learning of the unknown target function  $f \in \mathcal{C}$ . This means that they use independent uniform samples from  $f^{-1}(1)$  to construct a hypothesis function  $h \in \mathcal{C}$  which, with probability at least  $9/10$ , satisfies  $\text{rel-dist}(f, h) \leq \varepsilon/2$ .
2. Next, they output an  $\varepsilon/2$ -sampler for  $h$ .

We now show that relative-error learning algorithms for a class of functions yield relative-error testing algorithms for the same class, with query complexity comparable to the sample complexity of the learning algorithm:

**FACT 3.3.** *Let  $\mathcal{C}$  be a class of functions from  $\{0, 1\}^n$  to  $\{0, 1\}$ . Let  $A$  be an algorithm which performs  $\varepsilon$ -relative-error proper learning of  $\mathcal{C}$  using  $s(\varepsilon, n)$  uniform samples from  $f^{-1}(1)$ . Then there is an  $\varepsilon$ -relative-error testing algorithm  $T$  for  $\mathcal{C}$  which makes at most  $s(\varepsilon/4, n) + O(1/\varepsilon)$  calls to the sample oracle and at most  $O(1/\varepsilon)$  calls to the black-box oracle  $\text{MQ}(f)$ .*

*Proof.* The testing algorithm  $T$  works in the following stages:

1. Run the relative-error learning algorithm  $A$  with error parameter  $\varepsilon/4$ , using the sample oracle as the source of uniform random examples (this requires  $s(\varepsilon/4, n)$  calls to the sample oracle, and no calls to the  $\text{MQ}$  oracle). If  $A$  does not output a hypothesis  $h \in \mathcal{C}$  then output “reject.” Otherwise, let  $h \in \mathcal{C}$  denote the hypothesis that  $A$  outputs, and
2. Draw  $m := O(1/\varepsilon)$  samples from  $\text{Samp}(f)$  and evaluate  $h$  on each of them. If  $h$  evaluates to 0 on more than  $(3/8)\varepsilon m$  of the samples, then output “reject.” Otherwise,
3. Draw  $m$  independent uniform samples from  $h^{-1}(1)$ <sup>6</sup> and use  $\text{MQ}(f)$  to evaluate  $f$  on each of them. If  $f$  evaluates to 0 on more than  $(3/8)\varepsilon m$  of them then output “reject,” otherwise output “accept.”

The query complexity is clearly as claimed, so we turn to establishing correctness. Suppose first that the target function  $f$  belongs to  $\mathcal{C}$ . The probability that the  $(\varepsilon/4)$ -relative-error learning algorithm does not output a function  $h \in \mathcal{C}$  is at most  $1/10$ , so suppose that  $h \in \mathcal{C}$  is the output of the relative-error learning algorithm, and that  $\text{rel-dist}(f, h) \leq \varepsilon/4$ . Write

$$(3.1) \quad a \cdot 2^n := |f^{-1}(1) \cap h^{-1}(1)|, \quad b \cdot 2^n := |f^{-1}(1) \setminus h^{-1}(1)|, \quad c \cdot 2^n := |h^{-1}(1) \setminus f^{-1}(1)|.$$

Since  $\text{rel-dist}(f, h) = \frac{|f^{-1}(1) \triangle h^{-1}(1)|}{|f^{-1}(1)|} \leq \varepsilon/4$ , we have  $\frac{b+c}{a+b} \leq \varepsilon/4$ . Since the probability that  $h$  evaluates to 0 on a random example drawn from  $\text{Samp}(f)$  is  $\frac{b}{a+b} \leq \frac{b+c}{a+b} \leq \varepsilon/4$ , by a standard Chernoff bound the probability that  $T$  outputs “reject” in Step 2 is at most  $1/10$ ; so suppose that algorithm  $T$  proceeds to Step 3. Since the probability that  $f$  evaluates to 0 on a random example drawn from  $h^{-1}(1)$  is

$$\frac{c}{a+c} = \frac{a+b}{a+c} \cdot \frac{c}{a+b} \leq \frac{a+b}{a} \cdot \frac{\varepsilon}{4} = \frac{1}{1-b/(a+b)} \cdot \frac{\varepsilon}{4} \leq \frac{1}{1-\varepsilon/4} \cdot \frac{\varepsilon}{4} \leq \frac{\varepsilon}{3},$$

the probability that  $T$  outputs “reject” in Step 3 is at most  $1/10$ . So the overall probability that  $T$  outputs “reject” is at most  $1/10 + 1/10 + 1/10 < 1/3$ , as required since  $f \in \mathcal{C}$ .

Next, suppose that  $\text{rel-dist}(f, \mathcal{C}) > \varepsilon$ . If the relative-error learning algorithm does not output a hypothesis  $h \in \mathcal{C}$ , then the tester  $T$  outputs “reject” (as desired), so suppose that in stage 1 the tester  $T$  outputs a hypothesis  $h \in \mathcal{C}$ . Let  $a, b, c \in [0, 1]$  be as in Equation (3.1). Since  $\frac{b+c}{a+b} = \text{rel-dist}(f, h) \geq \text{rel-dist}(f, \mathcal{C}) > \varepsilon$ , it must be the case that either  $\frac{b}{a+b}$  (which is the fraction of examples in  $f^{-1}(1)$  that are labeled 0 by  $h$ ) is at least  $5\varepsilon/13$ , or  $\frac{c}{a+b}$  is at least  $8\varepsilon/13$ . In the first case, a Chernoff bound gives that  $T$  outputs “reject” in Step 2 with probability at least  $2/3$ , and in the second case,

$$\frac{c}{a+c} = \frac{1}{a/c+1} \geq \frac{1}{(a+b)/c+1} \geq \frac{1}{13/(8\varepsilon)+1} \geq \frac{8\varepsilon}{21}.$$

Note that  $\frac{c}{a+c}$  is the fraction of examples in  $h^{-1}(1)$  that are labeled 0 by  $f$ , so a Chernoff bound gives that  $T$  outputs “reject” in Step 3 with probability at least  $2/3$ .  $\square$

Fact 3.3 lets us obtain some positive results for relative-error testing of well-studied concept classes, namely LTFs and DNFs, from known positive results for learning those classes:

- En route to giving an efficient algorithm for learning the uniform distribution over satisfying assignments of an unknown LTF over  $\{0, 1\}^n$ , [DDS15] gives a  $\text{poly}(n, 1/\varepsilon)$ -sample algorithm which is an  $\varepsilon$ -relative-error proper learner for LTFs over  $\{0, 1\}^n$ . Hence by Fact 3.3, we get a  $\text{poly}(n, 1/\varepsilon)$ -query algorithm for relative-error testing of LTFs over  $\{0, 1\}^n$ .

<sup>6</sup>Note that while it may be a computationally hard task to generate a uniform random satisfying assignment of  $h$ , we are only concerned with the query complexity of our testing algorithm and not its running time.

- [DDS15] also gives a  $\text{poly}(n^{\log(s/\varepsilon)})$ -sample  $\varepsilon$ -relative-error proper learning algorithm for  $s$ -term DNFs over  $\{0, 1\}^n$  (as part of an algorithm for learning the uniform distribution over satisfying assignments of an unknown  $s$ -term DNF). Consequently, Fact 3.3 gives a  $\text{poly}(n^{\log(s/\varepsilon)})$ -query algorithm for  $\varepsilon$ -relative-error testing of the class of  $s$ -term DNFs.

On the other hand, Fact 3.3 is not useful for relative-error monotonicity testing, since even under the uniform distribution on  $\{0, 1\}^n$  (where both positive and negative random examples are available to the learner) no algorithm can learn monotone Boolean functions to accuracy  $\varepsilon$  using fewer than  $2^{\Omega(\sqrt{n}/\varepsilon)}$  samples [BCO<sup>+</sup>15].

#### 4 Algorithms for relative-error monotonicity testing

In this section we present algorithms for relative-error monotonicity testing. At a high level, our algorithms uses the same idea as the  $O(n/\varepsilon)$ -query edge tester for standard monotonicity testing. However, our algorithms will be much more efficient when the function being tested is *sparse*. In particular, let  $N$  be the sparsity of the given function  $f$ , i.e.  $N = |f^{-1}(1)|$ . Our algorithms make  $O(\log(N)/\varepsilon)$  calls to  $\text{MQ}(f)$  and  $\text{Samp}(f)$ , where  $\varepsilon$  is the (relative-error) distance parameter. Note that such an upper bound can still be highly non-trivial vis-a-vis standard monotonicity testing even when  $N = 2^{\Theta(n)}$ ; for example, if  $N = 2^{0.99n}$ , then having relative-error  $\varepsilon$  is a much stronger guarantee than having absolute error  $\varepsilon$ , since  $\text{ham-dist}(f, \mathcal{C}_{\text{monotone}}) = \text{rel-dist}(f, \mathcal{C}_{\text{monotone}}) \cdot 2^{-0.01n}$ .

We will present two versions of our algorithm that work in different settings. As a warmup, we first explain how a variant of the simple edge tester gives an  $O(n/\varepsilon)$ -query algorithm; this algorithm does not need to be provided with the value of  $N$ . Then, for our first main algorithm, we show how a modification of this simple algorithm works, using  $O(\log(N)/\varepsilon)$  queries, if the sparsity  $N$  is given to the algorithm as an input parameter. Finally, our second main algorithm does not require the sparsity  $N$  as an input parameter. It works for any Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , and as long as the function  $f$  has sparsity  $N$ , the algorithm will (usually) terminate within  $O(\log(N)/\varepsilon)$  steps. The idea of this algorithm is to first *learn* a rough estimate of the sparsity  $N$  and then run the first algorithm using the learned estimate of  $N$ . Because of this, the second algorithm is adaptive, and it terminates after  $O(\log(N)/\varepsilon)$  samples and queries only with high (say 0.99) probability. (We will show how to boost the probability to an arbitrarily large  $1 - \delta$  with only an  $O(\log(1/\delta)/\varepsilon)$  additive increase in the sample and query complexity.)

**4.1 Warmup: A nonadaptive  $O(n/\varepsilon)$ -query algorithm (that is not given  $N$ )** We first show that the  $O(n/\varepsilon)$  edge-tester algorithm in [GGL<sup>+</sup>00] still works in the relative-error setting with slight modification. The intuition of the edge-tester is simple: the algorithm just tries to find a *violating edge*, i.e. an edge  $\{x, y\}$  such that  $((x \prec y) \wedge (f(x) > f(y)))$ , and outputs “not monotone” if it finds such an edge. Since any far-from-monotone function must have “many” violating edges, an algorithm that samples some edges and outputs “not monotone” if and only if there is a violating edge among the sampled edges should work.

In more detail, we recall the following:

**LEMMA 4.1.** ([GGL<sup>+</sup>00, THEOREM 2]) *If  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  differs from any monotone function on at least  $\Delta$  points in  $\{0, 1\}^n$ , then the number of violating edges  $\{x, y\}$  for  $f$  is at least  $\Delta/2$ .*

**REMARK 5.** *The above lemma is tight up to constant factors [GGL<sup>+</sup>00, Proposition 4].*

If  $f$  satisfies  $\text{rel-dist}(f, \mathcal{C}_{\text{monotone}}) \geq \varepsilon$  and  $|f^{-1}(1)| = N$ , then  $f$  differs from any monotone function on at least  $\varepsilon N$  points. Thus, it directly follows from Lemma 4.1 that there are at least  $\varepsilon N/2$  violating edges for  $f$ . Moreover, all violating edges have one vertex  $x$  such that  $f(x) = 1$ , so in order to find violating edges, we can sample from the set of all edges that have at least one vertex  $x$  such that  $f(x)$ , and this can be done with the sampling oracle. We thus have the following algorithm:

##### Algorithm 1: An edge-testing algorithm

**Input:**  $n, \varepsilon$ , and access to the oracles  $\text{MQ}(f)$  and  $\text{Samp}(f)$ .

**Output:** Either “monotone” or “not monotone”.

1. Query the sample oracle  $\text{Samp}(f)$  for  $a := 4n/\varepsilon$  times.

2. For every sample  $x$  received, uniformly sample  $i \in [n]$ , and query the oracle  $\text{MQ}(f)$  on  $x \oplus e_i$  (which is  $x$  with its  $i$ -th bit flipped). If  $x_i = 0$  (so that  $x \prec x \oplus e_i$ ) and  $f(x \oplus e_i) = 0$  (which means  $\{x, x \oplus e_i\}$  is a violating edge), then output “not monotone” and halt.
3. If the algorithm did not output “not monotone” in Step 2, then output “monotone”.

The number of oracle calls made by the algorithm is  $2 \times 4n/\varepsilon = O(n/\varepsilon)$ . It is easy to see that the algorithm is non-adaptive, as the queries in Step 2 do not depend on one another and can thus be made in one round. Also, it is a one-sided algorithm, since when  $f$  is monotone, there is no violating edge, so the algorithm always outputs “monotone”.

Now consider the case where  $f$  is  $\varepsilon$ -relatively-far from monotone. Note that all violating edges are in the form of  $\{x, x \oplus e_i\}$  where  $f(x) = 1$ , so for each  $(x, i)$  uniformly sampled from  $f^{-1}(1) \times [n]$ , the probability that  $\{x, x \oplus e_i\}$  is a violating edge is at least  $(\varepsilon N/2)/(N \cdot n) = \varepsilon/(2n)$ , as there are at least  $\varepsilon N/2$  violating edges. Since the algorithm gets  $a$  uniform samples  $(x, i)$  from  $f^{-1}(1) \times [n]$ , the probability that none corresponds to a violating edge is at most

$$\left(1 - \frac{\varepsilon}{2n}\right)^a = \left(1 - \frac{\varepsilon}{2n}\right)^{4n/\varepsilon} \leq \frac{1}{3}$$

for sufficiently large  $n$ . So with probability at least  $2/3$ , the algorithm outputs “not monotone”.

**4.2 A non-adaptive  $O(\log(N)/\varepsilon)$ -query algorithm that is given  $N$**  In this section, we will improve the algorithm in the previous subsection and prove the following:

**THEOREM 4.1.** *There is a one-sided non-adaptive algorithm which, if it is given  $N := |f^{-1}(1)|$ , is an  $\varepsilon$ -relative-error tester for monotonicity making  $O(\log(N)/\varepsilon)$  calls to  $\text{Samp}(f)$  and  $\text{MQ}(f)$ .*

The key to this theorem is the following observation: if  $f$  is monotone and  $|f^{-1}(1)| = N$ , then for any  $x \in f^{-1}(1)$ , the value of  $\|x\|_1$  (i.e. the Hamming weight of  $x$ ) is at least  $n - \log_2 N$ , because  $f(y) = 1$  for any  $y \succ x$  and the number of such  $y$  is  $2^{n-\|x\|_1} \leq N$ . Therefore, if we know  $N$  and we found a point  $x \in \{0, 1\}^n$  with  $f(x) = 1$  and  $\|x\|_1 < n - \log_2 N$ , then  $f$  cannot be monotone.

Moreover, if we assume that every  $x$  we get from the sample oracle has Hamming weight close to  $n$ , then we can also improve the edge-testing algorithm (see Algorithm 1). Specifically, in the edge-testing algorithm, we sample violating edges by sample from all edges that have one vertex  $x$  in  $f^{-1}(1)$ . However, for  $x \in f^{-1}(1)$ , only edges going “upwards” from  $x$  (connecting  $x$  and  $x \oplus e_i$  where  $x \prec x \oplus e_i$ ) can be violating edges. Therefore, it suffices to sample only from edges going upwards from  $x$ , and moreover, the number of these edges is much smaller than  $n$  since  $x$  has Hamming weight close to  $n$ .

These observations naturally lead to the following algorithm.

**Algorithm 2:** A non-adaptive algorithm that is given  $N$

**Input:**  $n, \varepsilon, N$ , and access to the oracles  $\text{MQ}(f)$  and  $\text{Samp}(f)$ .

**Output:** Either “monotone” or “not monotone”.

1. Query the sample oracle  $\text{Samp}(f)$  for  $a := 16 \log_2(N)/\varepsilon$  times.
2. For every sample  $x$  received:
  - If  $\|x\|_1 < n - 2 \log_2 N$ , output “not monotone” and halt.
  - Otherwise, uniformly sample  $i$  from  $\{i \in [n] : x_i = 0\}$ , and query the oracle  $\text{MQ}(x)$  on  $x \oplus e_i$ . If  $f(x \oplus e_i) = 0$  (which means  $\{x, x \oplus e_i\}$  is a violating edge), then output “not monotone” and halt.
3. If the algorithm did not output “not monotone” in Step 2, then output “monotone”.

The number of oracle calls made by the algorithm is  $2 \times 16 \log_2(N)/\varepsilon = O(\log(N)/\varepsilon)$ . Similar to the previous subsection, the algorithm is non-adaptive, as the queries in Step 2 do not depend on one another and can thus be

made in one round, and the algorithm is also one-sided, since when  $f$  is monotone, there is no violating edge, so the algorithm always outputs “monotone”.

Now suppose that  $f$  is  $\varepsilon$ -relatively-far from monotone. Consider the following two cases:

- **Case 1:** There are at least  $\varepsilon N/2$  elements of  $f^{-1}(1)$  that have Hamming weight less than  $n - 2 \log_2 N$ . In this case, the probability that Algorithm 2 draws a sample that has Hamming weight less than  $n - 2 \log_2 N$  and outputs “not monotone” is at least

$$1 - \left(1 - \frac{\varepsilon N/2}{N}\right)^a = 1 - \left(1 - \frac{\varepsilon}{2}\right)^{16 \log_2(N)/\varepsilon} \geq 2/3.$$

- **Case 2:** There are at most  $\varepsilon N/2$  elements of  $f^{-1}(1)$  that has Hamming weight at least  $n - 2 \log_2 N$ . In this case, if one of the  $x$  sampled in Stage 1 of the algorithm has Hamming weight less than  $n - 2 \log_2 N$ , then the algorithm outputs “not monotone”. Below we condition on the event that all  $x$  sampled in the algorithm have Hamming weight at least  $n - 2 \log_2 N$ .

We first bound the number of violating edges both of whose vertices have Hamming weight at least  $n - 2 \log_2 N$ . Define function  $f' : \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $f'(t) = f(t)$  if  $\|t\|_1 \geq n - 2 \log_2 N$  and  $f'(t) = 0$  otherwise. Then  $\text{ham-dist}(f', f) < (\varepsilon N/2)/2^n$ , so by the triangle inequality, the number of points on which  $f'$  must disagree with any monotone function is greater than  $\varepsilon N/2$ . Therefore, by Lemma 4.1, there are at least  $\varepsilon N/4$  violating edges for  $f'$ , and thus at least  $\varepsilon N/4$  violating edges for  $f$  with both vertices having Hamming weight at least  $n - 2 \log_2 N$ .

For each violating edge both of whose vertices have Hamming weight at least  $n - 2 \log_2 N$ , the probability that the edge is  $\{x, x \oplus e_i\}$  when  $x$  is uniformly sampled from  $f^{-1}(1)$  and  $i$  is uniformly sampled such that  $x_i = 0$ , conditioning on  $\|x\|_1 \geq n - 2 \log_2 N$ , is at least  $1/(N \cdot 2 \log_2 N)$ . Since there are at least  $\varepsilon N/4$  such violating edges, the probability that Algorithm 2 encounters one of them and thus outputs “not monotone” is at least

$$1 - \left(1 - \frac{\varepsilon N/4}{2N \log_2 N}\right)^a = 1 - \left(1 - \frac{\varepsilon}{8 \log_2 N}\right)^{16 \log_2(N)/\varepsilon} \geq 2/3.$$

Hence, when  $f$  is  $\varepsilon$ -relatively-far from monotone, Algorithm 2 outputs “not monotone” with probability at least  $2/3$ .

**4.3 An  $O(\log(N)/\varepsilon)$ -complexity algorithm that does not know  $N$**  In this subsection, we consider the case when the algorithm does not have prior knowledge of  $N = |f^{-1}(1)|$ , and prove the following, which is a restatement of Theorem 1.1:

**THEOREM 1.1. (TESTING ALGORITHM)** *There is a one-sided adaptive algorithm which is an  $\varepsilon$ -relative-error tester for monotonicity (i.e., it always returns “monotone” when the input function  $f$  is monotone and returns “not monotone” with probability at least  $2/3$  when  $f$  has relative distance at least  $\varepsilon$  from monotone); with probability at least  $1 - \delta$ , it makes no more than  $O(\log(1/\delta)/\varepsilon + \log(N)/\varepsilon)$  calls to  $\text{Samp}(f)$  and  $\text{MQ}(f)$ .*

For this setting, the intuition is straight-forward: we first somehow get a rough estimate of  $N$ , and then proceed as in the previous section as if we knew  $N$ .

Let  $\hat{k}$  be the  $(\varepsilon N/2)$ -th-least Hamming weight among elements in  $f^{-1}(1)$ , that is, the  $(\varepsilon N/2)$ -th smallest item in (the multi-set)  $\{\|x\|_1 : x \in f^{-1}(1)\}$ . We will estimate  $\hat{k}$ , or more precisely, estimate a lower bound for  $\hat{k}$ , in the algorithm. Recall that for monotone  $f$  and  $x \in f^{-1}(1)$ , all  $y \succ x$  should satisfy  $f(y) = 1$ , so it is easy to see that  $N = |f^{-1}(1)| \geq 2^{n-\hat{k}}$ , and thus intuitively,  $\hat{k}$  gives useful information about the size of  $N$  for monotone  $f$ .

Now we describe the algorithm as in **Algorithm 3**. Let  $\delta \in (0, 1/2)$  be a parameter.

It is easy to see that the algorithm always outputs “monotone” when  $f$  is monotone given that it only returns “not monotone” when a violation to monotonicity is found. Below we consider the case when  $f$  is  $\varepsilon$ -relatively-far from monotone and show that the algorithm returns “not monotone” with probability at least  $2/3$ ; we analyze its query complexity at the end of the proof.

**Algorithm 3:** An algorithm that is not given  $N$ **Input:**  $n, \varepsilon$ , and access to the oracles  $\text{MQ}(f)$  and  $\text{Samp}(f)$ .**Output:** Either “monotone” or “not monotone”.

1. Query the sample oracle  $\text{Samp}(f)$  for  $c := 8 \log_2(1/\delta)/\varepsilon$  times. Let  $z$  be the sample that has the least Hamming weight (if there are multiple such  $z$ , choose an arbitrary one); set  $k := \|z\|_1$ . Uniformly sample  $y_1, \dots, y_b \in \{y \in \{0, 1\}^n : y \succ z\}$  where  $b := 4 \log_2(1/\delta)$ . If there exists  $j \in [b]$  such that  $f(y_j) = 0$ , output “not monotone” and halt.
2. Query the sample oracle  $\text{Samp}(f)$  for  $a := 16(n - k)/\varepsilon$  times.
3. For every sample  $x$  received:
  - Uniformly sample  $i$  from  $\{i \in [n] : x_i = 0\}$ , and query  $f(x \oplus e_i)$ . If  $f(x \oplus e_i) = 0$  (which means  $\{x, x \oplus e_i\}$  is a violating edge), output “not monotone” and halt.
4. If the algorithm has not output “not monotone” so far, output “monotone”.

We first show that at the end of Step 1, with probability at least  $\max\{5/6, 1 - \delta\}$ , either the algorithm outputs “not monotone” or  $k$  satisfies

$$(4.2) \quad n - 2 \log_2 N \leq k \leq \hat{k}.$$

Note that the number of elements in  $f^{-1}(1)$  that have Hamming distance at most  $\hat{k}$ , by the choice of  $\hat{k}$ , is at least  $\varepsilon N/2$ , so the probability that one of them appears as a sample in the first part of Step 1 (and thus  $k \leq \hat{k}$ ), is at least

$$1 - \left(1 - \frac{\varepsilon N/2}{N}\right)^c \geq 1 - \left(1 - \frac{\varepsilon}{2}\right)^{8 \log_2(1/\delta)/\varepsilon} \geq \max\left\{\frac{11}{12}, 1 - \frac{\delta}{2}\right\}.$$

On the other hand:

- If  $\|z\|_1 < n - 2 \log_2 N$  in Step 1, the probability that  $f(y_j) = 0$  for some  $j \in [b]$  is at least

$$1 - \left(\frac{N}{2^{2 \log_2 N}}\right)^b \geq 1 - \frac{1}{N^{4 \log_2(1/\delta)}} \geq \max\left\{\frac{11}{12}, 1 - \frac{\delta}{2}\right\},$$

in which case the algorithm will output “not monotone”.

Therefore, by a union bound, with probability at least  $\max\{5/6, 1 - \delta\}$ , either the algorithm outputs “not monotone” in Step 1, or  $k$  satisfies Equation (4.2).

Assuming Algorithm 3 does not output “not monotone” in Step 1 and  $k$  satisfies Equation (4.2), we show that with probability at least  $5/6$ , the algorithm outputs “not monotone” in Step 3.

To this end, we bound the probability of an  $x$  from  $\text{Samp}(f)$  leading Step 3 to return “not monotone”:

- Define  $f' : \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $f'(t) = f(t)$  if  $t$  has Hamming weight at least  $k$  and  $f'(t) = 0$  otherwise. Since  $k \leq \hat{k}$ , we have  $\text{ham-dist}(f', f) \leq \varepsilon N/2$ , so by triangle inequality, the Hamming distance between  $f'$  and any monotone function is at least  $\varepsilon N/2$ . Therefore, by Lemma 4.1, there are at least  $\varepsilon N/4$  violating edges for  $f'$ , and thus at least  $\varepsilon N/4$  violating edges for  $f$  with both vertices having Hamming weight at least  $k$ . As a result, when  $x$  is sampled from  $\text{Samp}(f)$ , the probability that Step 3 returns “not monotone” is at least

$$\frac{\varepsilon}{4} \cdot \frac{1}{n - k},$$

where the  $\varepsilon/4$  is the probability that  $x$  is in one of these  $\varepsilon N/4$  violating edges and  $1/(n - k)$  is the probability that the  $i$  sampled in Step 3 is exactly the direction of the edge that  $x$  lies in.

As a result, Step 3 returns “not monotone” with probability at least

$$1 - \left(1 - \frac{\varepsilon}{4(n-k)}\right)^a = 1 - \left(1 - \frac{\varepsilon}{4(n-k)}\right)^{16(n-k)/\varepsilon} \geq \frac{5}{6},$$

Therefore, when  $f$  is  $\varepsilon$ -relatively-far from monotone, Algorithm 3 outputs “not monotone” with probability at least  $2/3$ .

The number of oracle calls Algorithm 3 makes is at most

$$8 \log_2(1/\delta)/\varepsilon + 4 \log_2(1/\delta) + 2 \times 16(n-k)/\varepsilon = O(\log(1/\delta)/\varepsilon + \log(N)/\varepsilon)$$

when  $n - 2 \log_2 N \leq k$  holds from Equation (4.2), which happens with probability at least  $1 - \delta$ .

## 5 Warmup: A one-sided non-adaptive $\Omega(\log N)$ lower bound

As a warmup, we give a simple one-sided non-adaptive lower bound for relative-error monotonicity testing. In particular, we prove the following theorem, which is inspired by the one-sided non-adaptive monotonicity testing lower bound of [FLN<sup>+</sup>02] in the standard model:

**THEOREM 5.1.** *There is a constant  $\varepsilon_0 > 0$  and a parameter  $N = \Theta\left(\binom{n}{3n/4}\right)$  such that any one-sided nonadaptive algorithm for testing whether a function  $f$  with sparsity  $N$  is monotone or has relative distance at least  $\varepsilon_0$  from monotone functions must make  $\Omega(\log N)$  many queries.*

To prove this theorem, we focus on the following family of functions. For each  $i \in [n]$ , define  $f_i : \{0, 1\}^n \rightarrow \{0, 1\}$  as follows:

$$f_i(x) = \begin{cases} 1 & \text{if } \|x\|_1 > 3n/4 + 1 \\ 0 & \text{if } \|x\|_1 < 3n/4 \\ 1 - x_i & \text{if } \|x\|_1 \in \{3n/4, 3n/4 + 1\}. \end{cases}$$

Note that all  $f_i$  have the same sparsity  $N = \Theta\left(\binom{n}{3n/4}\right)$ .

It is easy to show that every  $f_i$  has large relative distance from monotone functions:

**CLAIM 5.1.** *For every  $i \in [n]$ , we have  $\text{rel-dist}(f_i, \mathcal{C}_{\text{monotone}}) = \Omega(1)$ .*

*Proof.* First observe that  $|f_i^{-1}(1)|$  can be upper bounded as

$$|f_i^{-1}(1)| \leq \sum_{j=3n/4}^n \binom{n}{j} = O\left(\binom{n}{3n/4}\right).$$

Define  $\mathcal{A} = \{x : x_i = 0 \text{ and } \|x\|_1 = 3n/4 - 1\}$  and  $\mathcal{A}^\uparrow = \{x : x_i = 1 \text{ and } \|x\|_1 = 3n/4\}$ . Observe that for each  $x \in \mathcal{A}$ , there is a unique  $y \in \mathcal{A}^\uparrow$  such that  $f_i(x) = 1$  and  $f_i(y) = 0$ . Further,  $x$  and  $y$  differ only in the  $i^{\text{th}}$  coordinate. This implies that  $\text{dist}(f_i, \mathcal{C}_{\text{monotone}}) \geq |\mathcal{A}|$ . Thus,

$$\text{rel-dist}(f_i, \mathcal{C}_{\text{monotone}}) \geq \Theta\left(\frac{|\mathcal{A}|}{\binom{n}{3n/4}}\right) = \Theta(1),$$

as was to be shown.  $\square$

Given Claim 5.1 (letting  $\varepsilon_0$  be the hidden constant), our goal is to show that any randomized, non-adaptive algorithm that can find a violation in  $f_i$  with probability at least  $2/3$ , for every  $i \in [n]$ , must make  $\Omega(n)$  queries. We start by showing that it suffices to prove the same  $\Omega(n)$  lower bound for any randomized, non-adaptive algorithm that has access to the black-box oracle  $\text{MQ}(f)$  only.

We say  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  is a *two-layer* function if

$$(5.3) \quad f(x) = \begin{cases} 1 & \text{if } \|x\|_1 > 3n/4 + 1 \\ 0 & \text{if } \|x\|_1 < 3n/4 \end{cases}$$

Note that every function  $f_i$  is a two-layer function as defined in Equation (5.3). Indeed all functions used in our lower bound proofs in the next two sections are two-layer functions as well.

One reason that two-layer testing is helpful for lower bound arguments is because the  $\text{Samp}(f)$  oracle is not needed for two-layer testing — it can be simulated, at low overhead, with the MQ oracle. This means that to prove a lower bound in our relative-error model for two-layer functions, it is sufficient to consider the slightly simpler setting in which the testing algorithm's only access to the unknown two-layer function  $f$  is via the MQ oracle.

**CLAIM 5.2.** *If  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  is a two-layer function, then  $q$  calls to the  $\text{Samp}(f)$  oracle can be simulated, with success probability at least  $99/100$ , by making  $O(q)$  calls to the MQ oracle.*

*Proof.* The simulation works by simply making  $Cq$  (for a suitable large constant  $C$ ) many  $\text{MQ}(f)$  queries on independent uniform random points  $\mathbf{x} \in \{0, 1\}^n$  that satisfy  $\|\mathbf{x}\|_1 \geq 3n/4$ , and using the first  $q$  points for which  $f(\mathbf{x}) = 1$ . It is clear that each such point  $\mathbf{x}$  in the sample (for which  $f(\mathbf{x}) = 1$ ) corresponds to an independent draw from  $\text{Samp}(f)$ ; correctness follows from a simple probabilistic argument using the fact that for any two-layer function  $f$ , at least an  $\Omega(1)$  fraction of all  $n$ -bit strings with Hamming weight at least  $3n/4$  are satisfying assignments of  $f$ .  $\square$

Given Claim 5.2, it suffices to show that any randomized, non-adaptive, MQ-only algorithm that can find a violation in  $f_i$  with probability at least  $(99/100) \cdot 2/3$  must make  $\Omega(n)$  MQ queries. To this end, letting  $\mathcal{D}_{\text{no}}$  be the uniform distribution on  $f_i$ , it suffices to prove Lemma 5.1 below, because any randomized,  $q$ -query, non-adaptive algorithm that can find a violation in every  $f_i$  with probability at least  $(99/100) \cdot 2/3$  implies a deterministic algorithm that finds a violation in  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  with probability at least  $(99/100) \cdot 2/3$ .

**LEMMA 5.1.** *Any deterministic,  $q$ -query, non-adaptive, MQ-only algorithm finds a violation in  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  with probability at most  $q/n$ .*

*Proof.* [Proof of Lemma 5.1] Let  $Q$  be the set of  $q$  queries. We assume without loss of generality that every  $x \in Q$  has  $\|x\|_1 \in \{3n/4, 3n/4 + 1\}$ . We will show that  $Q$  can reveal a violation for at most  $q - 1$  of the functions  $f_i$ . To argue this, consider an undirected graph  $G = (Q, E)$  in which the vertices correspond to queries in  $Q$ . Join any two vertices in  $Q$  with an edge labeled by  $j$  if they differ at exactly the  $j^{\text{th}}$  coordinate. Next, we drop all the edges until we just retain a spanning forest  $G$ . Let the edges of the spanning forest be  $E'$ . Note that the spanning forest has at most  $q - 1$  edges.

Now, consider a function  $f_i$  whose violation is revealed by the set of queries. For this to be the case, it must mean that there is a pair of adjacent vertices — call them  $u$  and  $v$  — in  $G$  such that the corresponding edge is labeled by  $i$ . This means that in the spanning forest,  $u$  and  $v$  must be in the same component. Furthermore, there must be at least one edge in the path connecting  $u$  and  $v$  in the spanning forest which is labeled by  $i$ . Since the number of edges in the spanning forest is at most  $q - 1$ , it follows that  $Q$  can reveal a violation for at most  $q - 1$  of the functions  $f_i$ .  $\square$

**REMARK 6.** *In Theorem 5.1 as well as the rest of the paper, we work with two-layer functions as described above, where the two layers in question are  $3n/4$  and  $3n/4 + 1$ . This choice is made just for concreteness to aid with readability; it is easy to verify that the definition of two-layer functions could be altered to use the two layers  $\alpha n$  and  $\alpha n + 1$ , for any constant  $\alpha \in (1/2, 1)$ , and that Theorem 5.1 would still go through with  $N = \Theta(\binom{n}{\alpha n})$ .*

## 6 A two-sided non-adaptive lower bound

The goal of this section is to prove the following two-sided, non-adaptive lower bound for relative-error monotonicity testing:

**THEOREM 6.1.** *Let  $N = \binom{n}{3n/4}$ . There is a constant  $\varepsilon_0 > 0$  such that any two-sided, non-adaptive algorithm for testing whether a function  $f$  with  $|f^{-1}(1)| = \Theta(N)$  is monotone or has relative distance at least  $\varepsilon_0$  from monotone functions must make  $\Omega(\log N)$  queries.*

To see that Theorem 6.1 implies Theorem 1.2, we first note that for any choice of the parameter  $N \leq \binom{n}{3n/4}$ , there exists a positive integer  $k \leq n$  such that  $N = \Theta(\binom{k}{3k/4})$ . The desired  $\Omega(\log N)$  lower bound for relative-error



testing of functions with sparsity  $\Theta(N)$  can then be obtained from a routine reduction to Theorem 6.1 (with  $n$  set to  $k$ ) by embedding in a suitable subcube of  $\{0, 1\}^n$  using functions  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  of the form

$$f(x_1, \dots, x_n) = (x_{k+1} \wedge \dots \wedge x_n) \wedge f'(x_1, \dots, x_k).$$

Moreover, as discussed in Definition 6,  $3/4$  can be replaced by any constant  $\alpha \in (1/2, 1)$ . Choosing  $\alpha$  to be sufficiently close to  $1/2$  extends the lower bound to any  $N \leq 2^{\alpha_0 n}$  for any constant  $\alpha_0 < 1$ .

As it will become clear in Section 6.1, all functions used in our lower bound proof are two-layer functions. We will prove that any randomized, non-adaptive, MQ-only algorithm for testing two-layer functions with success probability  $(99/100) \cdot 2/3$  must make  $\tilde{\Omega}(n)$  queries. It follows from Claim 5.2 that, even with access to both MQ and Samp, any randomized, non-adaptive algorithm for testing two-layer functions with success probability  $2/3$  must make  $\tilde{\Omega}(n)$  queries. In the rest of the section, we focus on MQ-only algorithms.

**6.1 Distributions  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$**  Our proof is an adaptation of the  $\tilde{\Omega}(\sqrt{n})$  lower bound from [CWX17] for two-sided non-adaptive monotonicity testing in the standard model.

To describe the construction of the yes- and no- distributions  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$ , we begin by describing the distribution  $\mathcal{E}$ .  $\mathcal{E}$  is uniform over all tuples  $T = (T_i : i \in [L])$ , where  $L := (4/3)^n$  and  $T_i : [n] \rightarrow [n]$ . Equivalently, to draw a tuple  $T \sim \mathcal{E}$ , for each  $i \in [L]$ , we sample a random  $T_i$  by sampling each  $T_i(k)$  independently and uniformly (with replacement) from  $[n]$  for each  $k \in [n]$ . We will refer to  $T_i$  as the  $i$ -th term in  $T$  and  $T_i(k)$  as the  $k$ -th variable of  $T_i$ . Given a term  $T_i : [n] \rightarrow [n]$ , we abuse the notation to use  $T_i$  to denote the Boolean function over  $\{0, 1\}^n$  with  $T_i(x) = 1$  if  $x_{T_i(k)} = 1$  for all  $k \in [n]$  and  $T_i(x) = 0$  otherwise. (So  $T_i$  is a conjunction, which is why we refer to it as a term as mentioned above.)

A function  $f \sim \mathcal{D}_{\text{yes}}$  is drawn using the following procedure:

1. Sample  $T \sim \mathcal{E}$  and use it to define the multiplexer map  $\Gamma = \Gamma_T : \{0, 1\}^n \rightarrow [L] \cup \{0^*, 1^*\}$ :

$$\Gamma_T(x) = \begin{cases} 0^* & T_i(x) = 0 \text{ for all } i \in [L] \\ 1^* & T_i(x) = 1 \text{ for at least two different } i \in [L] \\ i & T_i(x) = 1 \text{ for a unique } i \in [L]. \end{cases}$$

2. Sample  $H = (h_i : i \in [L])$  from  $\mathcal{E}_{\text{yes}}$ , where each  $h_i : \{0, 1\}^n \rightarrow \{0, 1\}$  is independently (1) with probability  $2/3$ , a random dictatorship Boolean function, i.e.,  $h_i(x) = x_k$  with  $k$  sampled uniformly at random from  $[n]$ ; and (2) with probability  $1/3$ , the all-0 function.
3. Finally, with  $T$  and  $H$  in hand,  $f = f_{T,H} : \{0, 1\}^n \rightarrow \{0, 1\}$  is defined as follows:  $f(x) = 1$  if  $\|x\|_1 > 3n/4 + 1$ ;  $f(x) = 0$  if  $\|x\|_1 < 3n/4$ ; if  $\|x\|_1 \in \{3n/4, 3n/4 + 1\}$ , we have

$$f(x) = \begin{cases} 0 & \Gamma_T(x) = 0^* \\ 1 & \Gamma_T(x) = 1^* \\ h_{\Gamma_T(x)}(x) & \text{otherwise (i.e., } \Gamma_T(x) \in [L]) \end{cases}$$

A function  $f \sim \mathcal{D}_{\text{no}}$  is drawn using the same procedure, with the exception that  $H = (h_i : i \in [L])$  is drawn from  $\mathcal{E}_{\text{no}}$  (instead of  $\mathcal{E}_{\text{yes}}$ ): Each  $h_i : \{0, 1\}^n \rightarrow \{0, 1\}$  is (1) with probability  $2/3$  a random anti-dictatorship Boolean function  $h_i(x) = \bar{x}_k$  with  $k$  drawn uniformly from  $[n]$ ; and (2) with probability  $1/3$ , the all-1 function. Note that every function  $f$  in the support of either  $\mathcal{D}_{\text{yes}}$  or  $\mathcal{D}_{\text{no}}$  is a two-layer function as defined in Equation (5.3).

Our construction of  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$  differs in various ways from the construction for the two-sided, non-adaptive lower bound in [CWX17], such as the number of terms and the biasing of the  $h_i$  functions in  $\mathcal{E}_{\text{yes}}$  and  $\mathcal{E}_{\text{no}}$ .

In the next two lemmas we show that functions in the support of  $\mathcal{D}_{\text{yes}}$  are monotone and  $f \sim \mathcal{D}_{\text{no}}$  is likely to have large relative distance from monotone functions.

**LEMMA 6.1.** *Every function in the support of  $\mathcal{D}_{\text{yes}}$  is monotone.*

*Proof.* Fix a  $T$  from the support of  $\mathcal{E}$  and  $H$  from the support of  $\mathcal{E}_{\text{yes}}$ . This fixes a function  $f$  in the support of  $\mathcal{D}_{\text{yes}}$ .

Note that for two-layer functions, the only nontrivial points lie on layers  $\{3n/4, 3n/4 + 1\}$ . Thus, consider any  $x \prec y$  such that  $\|x\|_1 = 3n/4$  and  $\|y\|_1 = 3n/4 + 1$ . Note that they in fact only differ on one bit. Assume that  $f(x) = 1$ , and we will show that  $f(y) = 1$ .

Since  $f(x) = 1$ , we know that  $x$  satisfies at least one term. Thus  $y$  satisfies at least one term. If  $y$  satisfies multiple terms, then  $f(y) = 1$ . So consider the case where  $y$  satisfies a unique term  $T_{i'}$ . Since  $f(x) = 1$ , it must be the case that  $x$  also uniquely satisfies the term  $T_{i'}$ , which implies  $h_{i'} \neq 0$ . Thus,  $h_{i'} = x_k$  for some  $k \in [n]$ . Since  $f(x) = 1$ , we have  $x_k = 1$ , which implies  $y_k = 1$  and  $f(y) = 1$ . This finishes the proof of the lemma.  $\square$

LEMMA 6.2. A function  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  satisfies  $\text{rel-dist}(\mathbf{f}, \mathcal{C}_{\text{monotone}}) = \Omega(1)$  with probability  $\Omega(1)$ .

*Proof.* We will show that with probability at least 0.0001, the number of disjoint violating pairs for monotonicity is  $\Omega(|\mathbf{f}^{-1}(1)|)$ . The lemma then follows directly from Lemma 4.1 and Definition 5.

Fix a function  $f$  from the support of  $\mathcal{D}_{\text{no}}$  (equivalently, fix  $T$  from the support of  $\mathcal{E}$  and  $H$  from the support of  $\mathcal{E}_{\text{no}}$ .) Note that violating pairs can only appear at the two non-trivial layers  $\{3n/4, 3n/4 + 1\}$ . Thus, at a high level, we want to show there are many disjoint pairs  $x \prec y$  such that 1)  $\|x\|_1 = 3n/4$  and  $\|y\|_1 = 3n/4 + 1$ , and 2)  $f(x) = 1$  and  $f(y) = 0$ .

Fix an  $x \in \{0, 1\}^n$  such that  $\|x\|_1 = 3n/4$ . Observe that if  $x$  is involved in some violating pair, then it must be the case that  $x$  satisfies a unique term  $T_{i'}$  for  $i' \in [L]$  and  $h_{i'} = \overline{x_k}$  for some  $k \in [n]$  such that  $x_k = 0$ . So the conditions above are necessary. It is also easy to see if they hold, then  $f(x) = 1$ , making  $x$  a candidate to be part of some violating pair.

Let  $y$  be the string that is obtained by flipping the  $k$ th bit of  $x$ . Ideally, we want to conclude that  $f(y) = 0$ . Indeed,  $h_{i'}(y) = 0$ , but we need to be careful to make sure that  $y$  still satisfies the unique term  $T_{i'}$ .

Formally, let  $X$  be the set of all points  $x$  such that 1)  $\|x\|_1 = 3n/4$ , 2)  $x$  satisfies a unique term  $T_{i'}$  for  $i' \in [L]$ , and  $h_{i'} = \overline{x_k}$  for some  $k \in [n]$  such that  $x_k = 0$ , 3)  $y$  (defined by flipping the  $k$ th bit of  $x$ ) still satisfies the unique term  $T_{i'}$ .

Note that every  $x \in X$  and the corresponding  $y$  form a disjoint violating pair (since they must satisfy the unique term so that the anti-dictatorship function is well-defined), thus the number of disjoint violating pairs is lower bounded by the size of  $X$ . Note that  $\mathbf{X}$  is a random variable depending on the choices of  $(\mathbf{T}, \mathbf{C})$  and  $\mathbf{H}$ . Next, we show that every fixed  $x$  with  $\|x\|_1 = 3n/4$  is in  $\mathbf{X}$  with constant probability.

CLAIM 6.1. For each  $x \in \{0, 1\}^n$  such that  $\|x\|_1 = 3n/4$ , we have

$$\Pr_{\mathbf{T} \sim \mathcal{E}, \mathbf{H} \in \mathcal{E}_{\text{no}}} [x \in \mathbf{X}] \geq 0.01.$$

*Proof.* We show the claim by calculating the probabilities in two steps. Fix  $x \in \{0, 1\}^n$  such that  $\|x\|_1 = 3n/4$  and  $k \in [n]$  such that  $x_k = 0$ . Let  $y$  be  $x$  but with the  $k$ -th coordinate changed to 1. Note that  $\|y\|_1 = 3n/4 + 1$ .

First, the probability that  $x$  and  $y$  uniquely satisfy a term  $\mathbf{T}_i$  for some  $i \in [L]$  is

$$L \cdot \left(\frac{3}{4}\right)^n \cdot \left(1 - \left(\frac{3}{4} + \frac{1}{n}\right)^n\right)^{L-1} \geq 1/e - 0.01 \geq 0.35,$$

for some sufficiently large  $n$ .

Fix such an  $i' \in [L]$ . The probability that  $\mathbf{h}_{i'} = \overline{x_k}$  is  $\frac{2}{3n}$ . Since  $x$  has  $n/4$  coordinates that are 0, we have

$$\Pr_{\mathbf{T} \sim \mathcal{E}, \mathbf{H} \in \mathcal{E}_{\text{no}}} [x \in \mathbf{X}] \geq 0.35 \cdot 1/6 \geq 0.05.$$

This finishes the proof of Claim 6.1.  $\square$

Note that the number of points on the  $3n/4$  layer is  $\binom{n}{3n/4} \geq |f^{-1}(1)|/100$ . Thus, the expected size of  $\mathbf{X}$  is at least  $|f^{-1}(1)|/10000$ . Thus, we have  $|\mathbf{X}| = \Omega(|f^{-1}(1)|)$  with probability at least 0.0001. This finishes the proof of Lemma 6.2.  $\square$

Let  $\varepsilon_0$  and  $c$  be the two constants hidden in the  $\Omega(1)$ 's in Lemma 6.2, i.e., a function  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  has relative distance to monotonicity at least  $\varepsilon_0$  with probability at least  $c$ . Let  $\alpha$  be the constant  $\min(\varepsilon_0, c)/100$ . Let

$q = n/(C_0 \log n)$  for some sufficiently large constant  $C_0$  to be specified later. We show that no randomized,  $q$ -query, non-adaptive, MQ-only algorithm can distinguish monotone functions from functions that have relative distance at least  $\varepsilon_0$  with probability at least  $1 - \alpha$ . This implies (using standard amplification techniques) that any randomized, non-adaptive, MQ-only algorithm that distinguishes the two cases with probability  $(99/100) \cdot 2/3$  must make  $\Omega(q)$  queries as desired.

To this end, we use Yao's minimax principle and consider a deterministic,  $q$ -query, non-adaptive, MQ-only algorithm ALG which consists of a set  $Q$  of  $q$  query strings  $x_1, \dots, x_q$  and a decision procedure which outputs "accept" or "reject" given  $f(x_k)$  for each  $k \in [q]$ . In order to prove the lower bound, it suffices to prove the following lemma:

LEMMA 6.3. *Let ALG be any non-adaptive, deterministic,  $q$ -query algorithm. Then*

$$\Pr_{f \sim \mathcal{D}_{\text{yes}}} [\text{ALG accepts } f] \leq (1 + \alpha) \cdot \Pr_{f \sim \mathcal{D}_{\text{no}}} [\text{ALG accepts } f] + \alpha.$$

Assume Lemma 6.3, and assume for a contradiction that there is a randomized, nonadaptive,  $q$ -query algorithm that can distinguish monotone functions from functions that have relative distance at least  $\varepsilon_0$  to  $\mathcal{C}_{\text{monotone}}$  with probability at least  $1 - \alpha$ . This implies a non-adaptive, deterministic algorithm ALG such that

$$\Pr_{f \sim \mathcal{D}_{\text{yes}}} [\text{ALG accepts } f] - \Pr_{f \sim \mathcal{D}_{\text{no}}} [\text{ALG accepts } f] \geq (1 - \alpha) - (1 - c(1 - \alpha)) \geq c/2,$$

using the choice of  $\alpha$ . This contradicts Lemma 6.3, again using the choice of  $\alpha$ .

**6.2 Proof of Lemma 6.3** Given that Lemma 6.3 is only about running ALG on  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$ , we may assume without loss of generality that every point queried by ALG lies in the two layers  $3n/4$  and  $3n/4 + 1$ . Given a function  $f = f_{T,H}$  from the support of either  $\mathcal{D}_{\text{yes}}$  or  $\mathcal{D}_{\text{no}}$  and a set  $Q$  of  $q$  queries, we define the following tuple  $(P; R; \rho)$  as the *outcome* of  $Q$  on  $f$ , where  $P = (P_i : i \in [L])$  with  $P_i \subseteq Q$ ,  $R = (R_i : i \in [L])$  with  $R_i \subseteq Q$ , and  $\rho = (\rho_i : i \in [L])$  with  $\rho_i : P_i \rightarrow \{0, 1\}$ :

1. Start with  $P_i = R_i = \emptyset$  for all  $i \in [L]$  (and  $\rho_i$  being the empty map given that  $P_i = \emptyset$ ).
2. We build  $P$  and  $R$  as follows. For each  $x \in Q$  (the order does not matter), if no term in  $T$  is satisfied, add  $x \rightarrow R_i$  for all  $i \in [L]$ ; if  $x$  satisfies a unique term in  $T$ , say  $T_i$ , add  $x \rightarrow P_i$  and  $x \rightarrow R_j$  for all  $j \neq i$ ; if  $x$  satisfies more than one term in  $T$ , letting  $i < i'$  be the first two terms satisfied by  $x$ , add  $x \rightarrow P_i$ ,  $x \rightarrow P_{i'}$ , and  $x \rightarrow R_j$  for all  $j : j < i'$  and  $j \neq i$ .
3. For each  $i \in [L]$  such that  $P_i \neq \emptyset$  and  $x \in P_i$ , set  $\rho_i(x) = h_i(x)$ .

We record the following facts about outcomes of  $Q$  on functions in the support of  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$ , which are evident by construction:

FACT 6.1. *The  $P$  and  $R$  parts of the outcome depend on  $T$  only. Once  $P$  is fixed, the  $\rho$  part of the outcome depends on  $H$  only.*

*Moreover, values of  $f(x)$ ,  $x \in Q$ , can be determined by the outcome of  $Q$  on  $f$ . As a result, if the outcomes of  $Q$  on two functions  $f$  and  $f'$  are the same, then ALG returns the same answer.*

FACT 6.2.  $\sum_{i \in [L]} |P_i| \leq 2q$ .

The following definition of bad vs good outcomes is crucial to the analysis:

DEFINITION 7. *Let  $(P; R; \rho)$  be a possible outcome of  $Q$  on a function  $f$  from the support of  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$ . We say  $(P; R; \rho)$  is a bad outcome if at least one of the following two events happen:*

- *For some  $i \in [L]$ , there exist  $x, y \in P_i$  such that*

$$|\{k \in [n] : x_k = y_k = 1\}| \leq 3n/4 - 100 \log n.$$

- *For some  $i \in [L]$ , there exist  $x, y \in P_i$  such that  $\rho_i(x) \neq \rho_i(y)$ .*

Otherwise, we say  $(P; R; \rho)$  is a good outcome.

Lemma 6.3 follows from the following two claims:

CLAIM 6.2. Let  $Q$  be a set of  $q$  queries. Then we have

$$\Pr_{T \sim \mathcal{E}, H \sim \mathcal{E}_{yes}} [\text{outcome of } Q \text{ on } f_{T,H} \text{ is bad}] \leq \alpha.$$

CLAIM 6.3. Let  $Q$  be a set of  $q$  queries. For any good outcome  $\ell = (P; R; \rho)$ , we have

$$\Pr_{T \sim \mathcal{E}, H \sim \mathcal{E}_{yes}} [\text{outcome of } Q \text{ on } f_{T,H} \text{ is } \ell] \leq (1 + \alpha) \cdot \Pr_{T \sim \mathcal{E}, H \sim \mathcal{E}_{no}} [\text{outcome of } Q \text{ on } f_{T,H} \text{ is } \ell].$$

*Proof.* [Proof of Lemma 6.3 assuming Claim 6.2 and Claim 6.3] By Fact 6.1 the outcome of  $Q$  on  $f$  uniquely determines  $f(x)$ ,  $x \in Q$ . This allows us to define  $L_1$  as the set of outcomes such that ALG accepts  $f$  if and only if the outcome of  $Q$  on  $f$  lies in  $L_1$ .

By Claim 6.2 and Claim 6.3, we have

$$\begin{aligned} \Pr_{f \sim \mathcal{D}_{yes}} [\text{ALG accepts } f] &= \sum_{\ell \in L_1} \Pr_{f \sim \mathcal{D}_{yes}} [\text{outcome of } Q \text{ on } f \text{ lies in } L_1] \\ &\leq \Pr_{f \sim \mathcal{D}_{yes}} [\text{outcome of } Q \text{ on } f \text{ is bad}] \\ &\quad + \sum_{\text{good } \ell \in L_1} \Pr_{f \sim \mathcal{D}_{yes}} [\text{outcome of } Q \text{ on } f \text{ is } \ell] \\ &\leq \alpha + \sum_{\text{good } \ell \in L_1} (1 + \alpha) \cdot \Pr_{f \sim \mathcal{D}_{no}} [\text{outcome of } Q \text{ on } f \text{ is } \ell] \\ &\leq (1 + \alpha) \cdot \Pr_{f \sim \mathcal{D}_{no}} [\text{ALG accepts } f] + \alpha. \end{aligned}$$

This finishes the proof of the lemma.  $\square$

We finish the proof by proving the two claims:

*Proof.* [Proof of Claim 6.3] Fix a good  $\ell = (P; R; \rho)$  and let  $I := \{i \in [L] : P_i \neq \emptyset\}$ . For each  $i \in I$ , let

$$A_{i,1} := \{k \in [n] : x_k = 1 \text{ for all } x \in P_i\} \quad \text{and} \quad A_{i,0} := \{k \in [n] : x_k = 0 \text{ for all } x \in P_i\}.$$

Since the outcome  $\ell$  is good, we have

- For all  $i \in [L]$  and  $x, y \in P_i$ , we have

$$|\{k \in [n] : x_k = y_k = 1\}| > 3n/4 - 100 \log n$$

and thus,

$$|\{k \in [n] : x_k = 1 \cap y_k = 0\}| = \|x\|_1 - |\{k \in [n] : x_k = y_k = 1\}| \leq 100 \log n.$$

For each  $i \in [L]$  with any  $x \in P_i$ , we have

$$3n/4 + 1 \geq \|x\|_1 \geq |A_{i,1}| \geq \|x\|_1 - \sum_{y \in P_i} |\{k \in [n] : x_k = 1 \cap y_k = 0\}| \geq 3n/4 - O(|P_i| \log n).$$

Similarly, we have

$$n/4 \geq n - \|x\|_1 \geq |A_{i,0}| \geq n - \|x\|_1 - \sum_{y \in P_i} |\{k \in [n] : x_k = 0 \cap y_k = 1\}| \geq n/4 - O(|P_i| \log n).$$

- For each  $i \in I$ , we use  $\rho_i$  to denote the value of  $\rho_i(x)$  shared by all  $x \in P_i$ .

Consider a fixed  $T$  in the support of  $\mathcal{E}$  such that the probability of  $(T, \mathbf{H})$  resulting in outcome  $\ell$  is positive when  $\mathbf{H} \sim \mathcal{E}_{\text{yes}}$ . Then it suffices to show that

$$\frac{\Pr_{\mathbf{H} \sim \mathcal{E}_{\text{no}}}[(T, \mathbf{H}) \text{ results in outcome } \ell]}{\Pr_{\mathbf{H} \sim \mathcal{E}_{\text{yes}}}[(T, \mathbf{H}) \text{ results in outcome } \ell]} \geq \frac{1}{1 + \alpha}.$$

By definition of the distributions  $\mathcal{E}_{\text{no}}$  and  $\mathcal{E}_{\text{yes}}$ , this ratio is a product over  $i \in I$ . For each  $i \in I$ , if  $\rho_i = 1$ , then the factor is

$$\frac{\frac{2}{3} \cdot \frac{|A_{i,0}|}{n} + \frac{1}{3}}{\frac{2}{3} \cdot \frac{|A_{i,1}|}{n}} = \frac{(2/3) \cdot |A_{i,0}| + (n/3)}{(2/3) \cdot |A_{i,1}|}.$$

Given that  $|A_{i,1}| = (3n/4) \pm O(|P_i| \log n)$  and  $|A_{i,0}| = (n/4) \pm O(|P_i| \log n)$  (and  $|P_i| \log n \leq 2q \log n$  which can be set to be sufficiently small compared to  $n$  by setting  $C_0$  to be sufficiently small),

$$\frac{(2/3) \cdot |A_{i,0}| + (n/3)}{(2/3) \cdot |A_{i,1}|} = \frac{(n/2) \pm O(|P_i| \log n)}{(n/2) \pm O(|P_i| \log n)} = 1 \pm O\left(\frac{|P_i| \log n}{n}\right).$$

Similarly when  $\rho_i = 0$ , the factor is

$$\frac{\frac{2}{3} \cdot \frac{|A_{i,1}|}{n}}{\frac{2}{3} \cdot \frac{|A_{i,0}|}{n} + \frac{1}{3}} = \frac{(2/3) \cdot |A_{i,1}|}{(2/3) \cdot |A_{i,0}| + (n/3)} = 1 \pm O\left(\frac{|P_i| \log n}{n}\right).$$

As a result, using  $\sum_{i \in I} |P_i| \leq 2q = 2n/(C_0 \log n)$  and setting  $C_0$  to be sufficiently small, we have

$$\frac{\Pr_{\mathbf{H} \sim \mathcal{E}_{\text{no}}}[(T, \mathbf{H}) \text{ results in outcome } \ell]}{\Pr_{\mathbf{H} \sim \mathcal{E}_{\text{yes}}}[(T, \mathbf{H}) \text{ results in outcome } \ell]} \geq \prod_{i \in I} \left(1 - O\left(\frac{|P_i| \log n}{n}\right)\right) \geq \frac{1}{1 + \alpha}.$$

This finishes the proof of the claim.  $\square$

*Proof.* [Proof of Claim 6.2] We first handle the first case of bad outcomes: some  $i \in [L]$  has  $x, y \in P_i$  with

$$|\{k \in [n] : x_k = y_k = 1\}| \leq (3n/4) - 100 \log n.$$

By a union bound over  $x, y$  and  $i$ , this happens with probability at most

$$(4/3)^n \cdot q^2 \cdot \left(\frac{(3n/4) - 100 \log n}{n}\right)^n = q^2 \left(1 - \frac{(4/3)100 \log n}{n}\right)^n = o(1).$$

Assume the first case does not happen. Define the same  $I$  and  $A_{i,0}, A_{i,1}$  as in the last proof. Using the same inequalities on  $|A_{i,0}|$  and  $|A_{i,1}|$ , we have for every  $i \in I$ ,

$$|[n] \setminus (A_{i,0} \cup A_{i,1})| \leq O(|P_i| \log n).$$

For the second case to happen, there must be an  $i \in I$  such that  $\mathbf{h}_i(x) = x_k$  sampled according to  $\mathcal{E}_{\text{yes}}$  has  $k \in [n] \setminus (A_{i,0} \cup A_{i,1})$ . Thus, taking a union bound over all possible  $i \in I$ , the probability over  $\mathbf{H} \sim \mathcal{E}_{\text{yes}}$  of resulting in such an outcome is at most

$$\sum_{i \in I} \left(\frac{2}{3} \cdot \frac{|[n] \setminus (A_{i,0} \cup A_{i,1})|}{n}\right) \leq \sum_{i \in I} \left(\frac{O(|P_i| \log n)}{n}\right) \leq \alpha,$$

by using  $\sum_{i \in I} |P_i| \leq 2q = 2n/(C_0 \log n)$  from Fact 6.2 and setting  $C_0$  to be sufficiently small.  $\square$

## 7 A two-sided adaptive lower bound

In this section we prove the following two-sided adaptive lower bound:

**THEOREM 7.1.** *Let  $N = \binom{n}{3n/4}$ . There is a constant  $\varepsilon_0 > 0$  such that any two-sided, adaptive algorithm for testing whether a function  $f$  with  $|f^{-1}(1)| = \Theta(N)$  is monotone or has relative distance at least  $\varepsilon_0$  from monotone functions must make  $\tilde{\Omega}((\log N)^{2/3})$  queries.*

Observations similar to those below the statement of Theorem 6.1 show that Theorem 7.1 implies Theorem 1.3. As it will become clear in Section 7.1, all functions used in our lower bound proof are two-layer functions as well. It follows from Claim 5.2 that it suffices to show that any randomized, adaptive, MQ-only algorithm for testing two-layer functions with success probability  $(99/100) \cdot 2/3$  must make  $\tilde{\Omega}((\log n)^{2/3})$  queries. So in the rest of the section, we focus on MQ-only algorithms.

**7.1 Distributions  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$**  Our proof is an adaptation of the  $\tilde{\Omega}(n^{1/3})$  lower bound from [CWX17] (see Section 3) for adaptive monotonicity testing in the standard model.

We start with the distribution  $\mathcal{E}$ . Let  $L := (4/3)^n$  and  $M := 4^n$ .  $\mathcal{E}$  is uniform over all pairs  $(T, C)$  of the following form:  $T = (T_i : i \in [L])$  with  $T_i : [n] \rightarrow [n]$  and  $C = (C_{i,j} : i \in [L], j \in [M])$  with  $C_{i,j} : [n] \rightarrow [n]$ . We call  $T_i$ 's the *terms* and  $C_{i,j}$ 's the *clauses*. Equivalently, to draw  $(T, C) \sim \mathcal{E}$ :

- For each  $i \in [L]$ , we sample a random term  $T_i$  by sampling  $T_i(k)$  independently and uniformly from  $[n]$  for each  $k \in [n]$ ;
- For each  $i \in [L]$  and  $j \in [M]$ , we sample a random clause  $C_{i,j}$  by sampling  $C_{i,j}(k)$  independently and uniformly from  $[n]$  for each  $k \in [n]$ .

Given a pair  $(T, C)$ , we interpret  $T_i$  as a term and abuse the notation to write

$$T_i(x) = \bigwedge_{k \in [n]} x_{T_i(k)}$$

as a Boolean function over  $\{0, 1\}^n$ . We say  $x$  satisfies  $T_i$  when  $T_i(x) = 1$ . On the other hand, we interpret each  $C_{i,j}$  as a clause and abuse the notation to write

$$C_{i,j}(x) = \bigvee_{k \in [n]} x_{C_{i,j}(k)}.$$

Each pair  $(T, C)$  defines a multiplexer map  $\Gamma = \Gamma_{T,C} : \{0, 1\}^n \rightarrow ([L] \times [M]) \cup \{0^*, 1^*\}$  as follows: We have  $\Gamma(x) = 0^*$  if  $T_i(x) = 0$  for all  $i \in [L]$  and  $\Gamma(x) = 1^*$  if  $T_i(x) = 1$  for at least two different  $i$ 's in  $[L]$ . Otherwise there is a unique  $i' \in [L]$  with  $T_{i'}(x) = 1$ . In this case the multiplexer enters the second level and examines  $C_{i',j}(x)$ ,  $j \in [M]$ . We have  $\Gamma(x) = 1^*$  if  $C_{i',j}(x) = 1$  for all  $j \in [M]$  and  $\Gamma(x) = 0^*$  if  $C_{i',j}(x) = 0$  for at least two different  $j$ 's in  $[M]$ . Otherwise there is a unique  $j' \in [M]$  such that  $C_{i',j'}(x) = 0$ , in which case the multiplexer outputs  $\Gamma(x) = (i', j')$ .

Next we define  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$ . A function  $\mathbf{f} \sim \mathcal{D}_{\text{yes}}$  is drawn using the following procedure:

1. Sample a pair  $(T, C) \sim \mathcal{E}$ , which is used to define a multiplexer map

$$\mathbf{\Gamma} = \mathbf{\Gamma}_{T,C} : \{0, 1\}^n \rightarrow ([L] \times [M]) \cup \{0^*, 1^*\}.$$

2. Sample  $\mathbf{H} = (\mathbf{h}_{i,j} : i \in [L], j \in [M])$  from  $\mathcal{E}_{\text{yes}}$ , where each  $\mathbf{h}_{i,j} : \{0, 1\}^n \rightarrow \{0, 1\}$  is (1) with probability  $2/3$ , a random dictatorship Boolean function, i.e.,  $\mathbf{h}_i(x) = x_k$  with  $k$  sampled uniformly at random from  $[n]$ ; and (2) with probability  $1/3$ , the all-0 function.
3. Finally,  $\mathbf{f} = \mathbf{f}_{T,C,H} : \{0, 1\}^n \rightarrow \{0, 1\}$  is defined as follows:  $\mathbf{f}(x) = 1$  if  $|x| > 3n/4 + 1$ ;  $\mathbf{f}(x) = 0$  if  $|x| < 3n/4$ ; and when  $|x| \in \{3n/4, 3n/4 + 1\}$ , we have

$$\mathbf{f}(x) = \begin{cases} 0 & \text{if } \mathbf{\Gamma}(x) = 0^* \\ 1 & \text{if } \mathbf{\Gamma}(x) = 1^* \\ \mathbf{h}_{\mathbf{\Gamma}(x)}(x) & \text{otherwise } (\mathbf{\Gamma}(x) \in [L] \times [M]) \end{cases}$$

A function  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  is drawn using the same procedure, with the exception that  $\mathbf{H} = (\mathbf{h}_{i,j} : i \in [L], j \in [M])$  is drawn from  $\mathcal{E}_{\text{no}}$  (instead of  $\mathcal{E}_{\text{yes}}$ ): Each  $\mathbf{h}_i : \{0, 1\}^n \rightarrow \{0, 1\}$  is (1) with probability  $2/3$  a random anti-dictatorship Boolean function  $\mathbf{h}_i(x) = \overline{x_k}$  with  $k$  drawn and uniformly from  $[n]$ ; and (2) with probability  $1/3$ , the all-1 function.

Our construction of  $\mathcal{D}_{\text{yes}}$  and  $\mathcal{D}_{\text{no}}$  again differs in various ways from the construction for the two-sided, adaptive lower bound in [CWX17], such as the number of terms and clauses and the biasing of  $\mathbf{h}_i$  in  $\mathcal{E}_{\text{yes}}$  and  $\mathcal{E}_{\text{no}}$ . (See Figure 2 and Figure 3 of [CWX17] for some illustrations that can be helpful in understanding the  $f_{T,C,H}$  functions.) Note that every function in the support of either  $\mathcal{D}_{\text{yes}}$  or  $\mathcal{D}_{\text{no}}$  is a two-layer function as defined in Equation (5.3).

In the next two lemmas we show that functions in the support of  $\mathcal{D}_{\text{yes}}$  are monotone and  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  is likely to have large relative distance from monotone functions.

**LEMMA 7.1.** *Every function  $f$  in the support of  $\mathcal{D}_{\text{yes}}$  is monotone.*

*Proof.* Fix a pair  $(T, C)$  from the support of  $\mathcal{E}$  and  $H$  from the support of  $\mathcal{E}_{\text{yes}}$ . This fixes a function  $f$  in the support of  $\mathcal{D}_{\text{yes}}$ .

Note that for two-layer functions, the only nontrivial points lie on layers  $\{3n/4, 3n/4 + 1\}$ . Thus, consider any  $x \prec y$  such that  $|x| = 3n/4$  and  $|y| = 3n/4 + 1$ . Note that they in fact only differ on one bit. Assume that  $f(x) = 1$ , and we will show that  $f(y) = 1$ .

Since  $f(x) = 1$ , we know that  $x$  satisfies at least one term and falsifies at most one clause. This implies that  $y$  satisfies at least one term and falsifies at most one clause as well. If  $y$  satisfies multiple terms *or* satisfies a unique term  $T_{i'}$  and falsifies no clauses, then  $f(y) = 1$ . So consider the case where  $y$  satisfies a unique term  $T_{i'}$  and falsifies a unique clause  $C_{i',j'}$ . Since  $x$  is below  $y$  and  $f(x) = 1$ , we have  $x$  uniquely satisfies the term  $T_{i'}$  and uniquely falsifies the clause  $C_{i',j'}$  as well. Again, since  $f(x) = 1$ , we have  $h_{i',j'} \neq 0$ .

This means  $h_{i',j'} = x_k$  for some  $k \in [n]$ . Since  $f(x) = 1$ , we have  $x_k = 1$ , which implies  $y_k = 1$  and  $f(y) = 1$ . This finishes the proof.  $\square$

**LEMMA 7.2.** *A function  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  satisfies  $\text{rel-dist}(\mathbf{f}, \mathcal{C}_{\text{monotone}}) = \Omega(1)$  with probability  $\Omega(1)$ .*

*Proof.* We will show that with probability at least 0.0001, the number of disjoint violating pairs to monotonicity is  $\Omega(|\mathbf{f}^{-1}(1)|)$ . The lemma then follows directly from Lemma 4.1.

Fix a function  $f$  from the support of  $\mathcal{D}_{\text{no}}$  (equivalently, fix  $(T, C)$  from the support of  $\mathcal{E}$  and  $H$  from the support of  $\mathcal{E}_{\text{no}}$ .) Note that violating pairs can only appear at the two non-trivial layers  $\{3n/4, 3n/4 + 1\}$ . Thus, at a high level, we want to show there are many disjoint pairs  $x \prec y$  such that 1)  $|x| = 3n/4$  and  $|y| = 3n/4 + 1$ , and 2)  $f(x) = 1$  and  $f(y) = 0$ .

Fix an  $x \in \{0, 1\}^n$  such that  $|x| = 3n/4$ . Observe that if  $x$  is involved in some violating pair, then it must be the case that  $x$  satisfies a unique term  $T_{i'}$  for  $i' \in [L]$  and falsifies a unique clause  $C_{i',j'}$  for  $j' \in [M]$ . Furthermore, it needs to be that  $h_{i',j'} = \overline{x_k}$  for some  $k \in [n]$  such that  $x_k = 0$ . So the conditions above are necessary. It's also easy to see if they hold, then  $f(x) = 1$ , making  $x$  a candidate point to be part of some violating pair.

Let  $y$  be the string that is obtained by flipping the  $k$ th bit of  $x$ . Ideally, we want to conclude that  $f(y) = 0$ . Indeed,  $h_{i',j'}(y) = 0$ , but we need to be careful to make sure that  $y$  still satisfies the unique term  $T_{i'}$  and falsifies the unique clause  $C_{i',j'}$ .

Formally, let  $X$  be the set of points  $x$  such that 1)  $|x| = 3n/4$ , 2)  $x$  satisfies a unique term  $T_{i'}$  for  $i' \in [L]$ , falsifies a unique clause  $C_{i',j'}$  for  $j' \in [M]$ , and  $h_{i',j'} = \overline{x_k}$  for some  $k \in [n]$  such that  $x_k = 0$ , 3)  $y$  (defined by flipping the  $k$ th bit of  $x$  from 0 to 1) satisfies the unique term  $T_{i'}$  and falsifies the unique clause  $C_{i',j'}$ .

Note that every  $x$  and the corresponding  $y$  form a disjoint violating pair (since they must satisfy the unique term and clause so that the anti-dictatorship function is well-defined), thus the number of disjoint violating pairs is lower bounded by the size of  $X$ . Note that  $\mathbf{X}$  is a random variable depending on the choices of  $(T, C)$  and  $\mathbf{H}$ . Next, we show that every fixed  $x$  is in  $\mathbf{X}$  with constant probability:

**CLAIM 7.1.** *For each  $x \in \{0, 1\}^n$  such that  $|x| = 3n/4$ , we have*

$$\Pr_{(T,C) \sim \mathcal{E}, \mathbf{H} \in \mathcal{E}_{\text{no}}} [x \in \mathbf{X}] \geq 0.01.$$

*Proof.* We establish the claim by calculating the probability in three steps. Fix  $x \in \{0, 1\}^n$  such that  $|x| = 3n/4$  and  $k \in [n]$  such that  $x_k = 0$ . Let  $y$  be  $x$  but with the  $k$ th coordinate flipped to 1. Note that  $|y| = 3n/4 + 1$ .

First, the probability that  $x$  and  $y$  uniquely satisfy a term  $T_i$  for some  $i \in [L]$  is

$$L \cdot \left(\frac{3}{4}\right)^n \cdot \left(1 - \left(\frac{3}{4} + \frac{1}{n}\right)^n\right)^{L-1} \geq 1/e - 0.01 \geq 0.35,$$

for some sufficiently large  $n$ .

Fix such an  $i' \in [L]$ . Second, the probability that  $x$  and  $y$  uniquely falsify a clause  $C_{i',j}$  for some  $j \in [M]$  is

$$M \cdot \left(\frac{1}{4} - \frac{1}{n}\right)^n \cdot \left(1 - \left(\frac{1}{4}\right)^n\right)^{M-1} \geq 1/e - 0.01 \geq 0.35,$$

for some sufficiently large  $n$ .

Fix such a  $j' \in [M]$ . The probability that  $\mathbf{h}_{i',j'} = \overline{x_k}$  is  $\frac{2}{3n}$ . Since  $x$  has  $n/4$  coordinates that are 0, we have

$$\Pr_{(\mathbf{T}, \mathbf{C}) \sim \mathcal{E}, \mathbf{H} \in \mathcal{E}_{\text{no}}} [x \in \mathbf{X}] \geq (0.35)^2 \cdot 1/6 \geq 0.02.$$

This finishes the proof.  $\square$

Note that the number of points on the  $3n/4$  layer is  $\binom{n}{3n/4} \geq |f^{-1}(1)|/100$ . Thus, the expected size of  $\mathbf{X}$  is at least  $|f^{-1}(1)|/10000$ , and  $|\mathbf{X}| = \Omega(|f^{-1}(1)|)$  with probability at least 0.0001.  $\square$

Let  $\varepsilon_0$  and  $c$  be the two constants hidden in the  $\Omega(1)$ 's in Lemma 7.2, i.e., a function  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  has relative distance to monotonicity at least  $\varepsilon_0$  with probability at least  $c$ . Let  $\alpha = \min(\varepsilon_0, c)/100$  and  $q := (n/C_0 \log n)^{2/3}$  for some sufficiently large constant  $C_0$  to be specified later. Following arguments similar to those around Lemma 6.3 in the previous section, it suffice to prove the following lemma for deterministic, adaptive,  $q$ -query, MQ-only algorithms.

LEMMA 7.3. *Let ALG be any deterministic, adaptive,  $q$ -query, MQ-only algorithm. Then we have*

$$\Pr_{(\mathbf{T}, \mathbf{C}) \sim \mathcal{E}, \mathbf{H} \sim \mathcal{E}_{\text{yes}}} [\text{ALG accepts } f_{\mathbf{T}, \mathbf{C}, \mathbf{H}}] \leq (1 + \alpha) \cdot \Pr_{(\mathbf{T}, \mathbf{C}) \sim \mathcal{E}, \mathbf{H} \sim \mathcal{E}_{\text{no}}} [\text{ALG accepts } f_{\mathbf{T}, \mathbf{C}, \mathbf{H}}] + \alpha.$$

**7.2 Proof of Lemma 7.3** Now we start the proof of Lemma 7.3. Formally, a deterministic, adaptive,  $q$ -query algorithm ALG is a depth- $q$  binary decision tree, in which each internal node  $u$  is labelled with a query point  $x_u$  and has two children with edges to them labelled with 0 and 1, respectively. Each leaf of the tree is labelled either “accept” or “reject.” Given any function  $f$ , the execution of ALG on  $f$  induces a path in the tree and ALG returns the label of the leaf at the end. Since the goal of ALG is to distinguish  $\mathcal{D}_{\text{yes}}$  from  $\mathcal{D}_{\text{no}}$  (as described in the statement of Lemma 7.3), we may assume without loss of generality that every point queried in ALG lies in the two layers of  $3n/4$  and  $3n/4 + 1$ .

Given a function  $f = f_{\mathbf{T}, \mathbf{C}, \mathbf{H}}$  from the support of either  $\mathcal{D}_{\text{yes}}$  or  $\mathcal{D}_{\text{no}}$ , we define the *outcome* of ALG on  $f$  to be the following tuple  $(Q; P; R; \rho)$ , where  $Q$  is the set of  $\leq q$  queries made by ALG along the path,  $P = (P_i, P_{i,j} : i \in [L], j \in [M])$ ,  $R = (R_i, R_{i,j} : i \in [L], j \in [M])$  with every  $P_i, P_{i,j}, R_i, R_{i,j} \subseteq Q$ , and  $\rho = (\rho_{i,j} : i \in [L], j \in [M])$  with  $\rho_{i,j} : P_{i,j} \rightarrow \{0, 1\}$ .  $P, R, \rho$  are built as follows:

1. Start with  $P_i = R_i = P_{i,j} = R_{i,j} = \emptyset$  for all  $i \in [L]$  and  $j \in [M]$ .
2. We build  $P$  and  $R$  as follows. For each query  $x \in Q$  made along the path, if no term in  $T$  is satisfied, add  $x \rightarrow R_i$  for all  $i \in [L]$ ; if  $x$  satisfies more than one term in  $T$ , letting  $i < i'$  be the first two terms satisfied by  $x$ , add  $x \rightarrow P_i, x \rightarrow P_{i'}$ , and  $x \rightarrow R_k$  for all  $k : k < i'$  and  $k \neq i$ ; if  $x$  satisfies a unique term in  $T$ , say  $T_i$ , add  $x \rightarrow P_i$  and  $x \rightarrow R_k$  for all  $k \neq i$ . For the last case we examine clauses  $C_{i,j}, j \in [M]$ , on  $x$ . If no clause  $C_{i,j}, j \in [M]$ , is falsified by  $x$ , add  $x \rightarrow R_{i,j}$  for all  $j \in [M]$ ; if more than one clause  $C_{i,j}, j \in [M]$ , are falsified by  $x$ , letting  $j < j'$  be the first two such clauses, then add  $x \rightarrow P_{i,j}, x \rightarrow P_{i,j'}$  and  $x \rightarrow R_{i,k}$  for all  $k : k < j'$  and  $k \neq j$ ; if a unique clause  $C_{i,j}$  is falsified by  $x$  for some  $j \in [M]$ , add  $x \rightarrow P_{i,j}$  and  $x \rightarrow R_{i,k}$  for all  $k \neq j$ .
3. For each  $i \in [L], j \in [M]$  and  $x \in P_{i,j}$ , set  $\rho_{i,j}(x) = h_{i,j}(x)$ . (So  $\rho_{i,j}$  is the dummy empty map for any  $i, j$  with  $P_{i,j} = \emptyset$ .)



We record the following facts about outcomes of ALG:

FACT 7.1. For each  $x \in Q$ , the value of  $f(x)$  is uniquely determined by the outcome of ALG on  $f$ . So if the outcomes of ALG on two functions  $f$  and  $f'$  are the same, then ALG returns the same answer on all queries in  $Q$ .

FACT 7.2. For each  $i \in [L]$ , we have  $\cup_{j \in [M]} P_{i,j} \subseteq P_i$ . We also have

$$\sum_{i \in [L]} |P_i| \leq 2|Q| \quad \text{and} \quad \sum_{j \in [M]} |P_{i,j}| \leq 2|P_i|, \quad \text{for every } i \in [L].$$

We say the outcome  $(Q; P; R; \rho)$  of ALG on  $f$  (from the support of either  $\mathcal{D}_{\text{yes}}$  or  $\mathcal{D}_{\text{no}}$ ) is *good* if it satisfies the following two conditions:

1. Condition  $C_1$ : For every  $i \in [L]$  and  $j \in [M]$  with  $P_{i,j} \neq \emptyset$ , letting

$$A_{i,j,0} = \{k \in [n] : x_k = 1 \text{ for all } x \in P_{i,j}\} \quad \text{and} \quad A_{i,j,1} = \{k \in [n] : x_k = 0 \text{ for all } x \in P_{i,j}\},$$

we have

$$(7.4) \quad |A_{i,j,0}| - n/4, |A_{i,j,1}| - 3n/4 \leq O\left(\min\{|P_{i,j}|^2, |P_i|\} \cdot \log n\right).$$

2. Condition  $C_2$ : For every  $i, j$  with  $P_{i,j} \neq \emptyset$ , we have  $\rho_{i,j}(x) = \rho_{i,j}(y)$  for all  $x, y \in P_{i,j}$ .

Similar to Lemma 6.3 in the previous section, Lemma 7.3 follows from the next two claims:

CLAIM 7.2. Let  $\ell = (Q; P; R; \rho)$  be a good outcome. Then we have

$$\begin{aligned} \Pr_{(T,C) \sim \mathcal{E}, H \sim \mathcal{E}_{\text{yes}}} [\text{outcome of ALG on } f_{T,C,H} \text{ is } \ell] \\ \leq (1 + \alpha) \cdot \Pr_{(T,C) \sim \mathcal{E}, H \sim \mathcal{E}_{\text{no}}} [\text{outcome of ALG on } f_{T,C,H} \text{ is } \ell]. \end{aligned}$$

*Proof.* The proof is similar to that of Claim 6.3.

Fix a good outcome  $\ell = (Q; P; R; \rho)$ . Consider a fixed pair  $(T, C)$  in the support of  $\mathcal{E}$  such that the probability of  $(T, C, H)$  resulting in  $\ell$  is positive when  $H \sim \mathcal{E}_{\text{yes}}$ . Then it suffices to show that

$$(7.5) \quad \frac{\Pr_{H \sim \mathcal{E}_{\text{no}}}[(T, C, H) \text{ results in outcome } \ell]}{\Pr_{H \sim \mathcal{E}_{\text{yes}}}[(T, C, H) \text{ results in outcome } \ell]} \geq \frac{1}{1 + \alpha}.$$

Let  $I$  be the set of  $i \in [L]$  such that  $P_i \neq \emptyset$ ; for each  $i \in [L]$ , let  $J_i$  be the set of  $j \in [M]$  such that  $P_{i,j} \neq \emptyset$ . Using (7.4) and an argument similar to that used in Claim 6.3, this ratio is at least

$$1 - \frac{1}{n} \cdot \sum_{i \in I} \sum_{j \in J_i} O\left(\min\{|P_{i,j}|^2, |P_i|\} \cdot \log n\right).$$

As  $\sum_{j \in J_i} |P_{i,j}| \leq 2|P_i|$  by Fact 7.2, we have that

$$\sum_{j \in J_i} \min\{|P_{i,j}|^2, |P_i|\}$$

is maximized when  $|J_i| = 2\sqrt{|P_i|}$  and  $|P_{i,j}| = \sqrt{|P_i|}$ , in which case

$$\sum_{i \in I} \sum_{j \in J_i} \min\{|P_{i,j}|^2, |P_i|\} = \sum_{i \in I} 2|P_i|^{3/2} \leq O(q^{3/2})$$

since  $\sum_{i \in I} |P_i| \leq 2q$ . (7.5) follows from the choice of  $q$  and by setting  $C_0$  to be sufficiently large.  $\square$

CLAIM 7.3. We have

$$\Pr_{(T,C) \sim \mathcal{E}, H \sim \mathcal{E}_{\text{yes}}} [\text{outcome of ALG on } f_{T,C,H} \text{ is good}] \geq 1 - \alpha.$$

We prove Claim 7.3 in Section 7.3. Lemma 7.3 follows from Claim 7.2 and Claim 7.3, with a proof similar to that of Lemma 6.3.

**7.3 Proof of Claim 7.3** First we describe the following equivalent view of running ALG on a function  $f$  from the support of  $\mathcal{D}_{\text{yes}}$ : After each query, ALG receives the outcome  $(Q; P; R; \rho)$  of queries made so far, where  $Q$  is the set of queries made along the path so far and  $P, R, \rho$  are built similar to before using  $Q$ . From the outcome  $(Q; P; R; \rho)$ , ALG can uniquely recover (Fact 7.1)  $f(x)$  for all queries made so far and for the most recent query in particular; it then uses the latter to walk down the tree to make the next query, and repeats this until a leaf is reached.

With this view, we describe algorithm  $\text{ALG}'$  obtained by modifying ALG as follows: After querying  $x$ , the outcome  $(Q; P; R; \rho)$  that ALG receives is updated differently as follows:

1.  $x$  is added to  $Q$ , some sets of  $P$  and some sets of  $R$  in the same way as before.
2. When  $x$  is added to a  $P_{i,j}$  that is nonempty before  $x$  is queried, instead of setting  $\rho_{i,j}(x)$  to be  $h_{i,j}(x)$ ,  $\rho_{i,j}(x)$  is set to be the value of  $\rho_{i,j}(y)$  of any  $y \in P_{i,j}$  that was added to  $P_{i,j}$  before  $x$ . (Note that if  $x$  is the first point added to  $P_{i,j}$ , then  $\rho_{i,j}(x)$  is set to be  $h_{i,j}(x)$  as before.)

Once  $\text{ALG}'$  receives the updated outcome, it recovers from it the value of  $f$  at  $x$  (which could be different from the real value of  $f(x)$ ) and simulates ALG for one step to make the next query. Note that this modification ensures that at any time, the outcome of queries made so far as seen by  $\text{ALG}'$  always satisfies Condition  $C_2$ .

We prove the following two claims, from which Claim 7.3 follows. We prove Claim 7.5 first, and then prove Claim 7.4 in the rest of the section.

CLAIM 7.4. *The outcome of  $\text{ALG}'$  on  $\mathbf{f} \sim \mathcal{D}_{\text{yes}}$  satisfies Condition  $C_1$  with probability  $1 - o(1)$ .*

CLAIM 7.5. *For any fixed outcome  $\ell$  of  $\text{ALG}'$  on  $\mathbf{f} \sim \mathcal{D}_{\text{yes}}$  that satisfies Condition  $C_1$ , we have*

$$\Pr_{\mathbf{f} \sim \mathcal{D}_{\text{yes}}} [\text{outcome of ALG on } \mathbf{f} \text{ is } \ell] \geq (1 - \alpha/2) \cdot \Pr_{\mathbf{f} \sim \mathcal{D}_{\text{yes}}} [\text{outcome of ALG}' on } \mathbf{f} \text{ is } \ell].$$

*Proof.* Fix an outcome  $\ell = (Q; P; R; \rho)$  of  $\text{ALG}'$  that satisfies Condition  $C_1$ . For each  $P_{i,j}$  that is not empty, we let  $x^{(i,j)}$  denote the first point that is added to  $P_{i,j}$ . Let  $\rho_{i,j} = \rho_{i,j}(x^{(i,j)})$ , which by the description of  $\text{ALG}'$  is the same as  $h_{i,j}(x^{(i,j)})$ .

To prove the claim, we show that for any fixed  $(T, C)$  such that the probability that the outcome of  $\text{ALG}'$  on  $\mathbf{f} = f_{T,C,\mathbf{H}}$  with  $\mathbf{H} \sim \mathcal{E}_{\text{yes}}$  is positive, we have

$$\frac{\Pr_{\mathbf{H} \sim \mathcal{D}_{\text{yes}}} [\text{outcome of ALG on } f_{T,C,\mathbf{H}} \text{ is } \ell]}{\Pr_{\mathbf{H} \sim \mathcal{D}_{\text{yes}}} [\text{outcome of ALG}' on } f_{T,C,\mathbf{H}} \text{ is } \ell]} \geq 1 - \frac{\alpha}{2}.$$

Similar to arguments used before, this ratio can be written as a product of factors, one for each  $i, j$  with  $P_{i,j} \neq \emptyset$ . For any  $i, j$  with  $P_{i,j} \neq \emptyset$  and  $\rho_{i,j} = 0$ , the factor is

$$\frac{\frac{1}{3} + \frac{2}{3} \cdot \frac{|A_{i,j,0}|}{n}}{\frac{1}{3} + \frac{2}{3} \cdot \frac{n - |x^{(i,j)}|}{n}} \geq 1 - \frac{1}{n} \cdot O(\min\{|P_{i,j}|^2, |P_i|\} \log n).$$

using the assumption that  $\ell$  satisfies Condition  $C_1$ , where  $A_{i,j,0}$  as before is the set of  $k \in [n]$  with  $x_k = 0$  for all  $x \in P_{i,j}$ . Similarly, when  $\rho_{i,j} = 1$  the factor is

$$\frac{\frac{2}{3} \cdot \frac{|A_{i,j,1}|}{n}}{\frac{2}{3} \cdot \frac{|x^{(i,j)}|}{n}} \geq 1 - \frac{1}{n} \cdot O(\min\{|P_{i,j}|^2, |P_i|\} \log n),$$

again using Condition  $C_1$ . The rest of the proof follows from calculations similar to those at the end of the proof of Claim 7.2.  $\square$

To prove Claim 7.4, we start with a sufficient condition for the outcome of  $\text{ALG}'$  to satisfy Condition  $C_1$ . We say the execution of  $\text{ALG}'$  on  $f$  from the support of  $\mathcal{D}_{\text{yes}}$  is *regular* if the following condition holds in every round: Let  $(Q; P; R; \rho)$  be the outcome of queries revealed to  $\text{ALG}'$  so far (recall the modification of  $\rho$  since we are interested in  $\text{ALG}'$  instead of ALG), let  $x$  be the next query, and let  $(Q'; P'; R'; \rho')$  be the updated outcome after  $x$  is queried. Then they satisfy

- For every  $i \in [L]$  with  $P_i \neq \emptyset$ ,  $|A_{i,1} \setminus A'_{i,1}| \leq 100 \log n$ , where

$$A_{i,1} := \{k \in [n] : y_k = 1 \text{ for all } y \in P_i\} \quad \text{and} \quad A_{i,1'} := \{k \in [n] : y_k = 1 \text{ for all } y \in P'_i\}.$$

- For every  $i \in [L]$ ,  $j \in [M]$  with  $P_{i,j} \neq \emptyset$ ,  $|A_{i,j,0} \setminus A'_{i,j,0}| \leq 100 \log n$ , where

$$A_{i,j,0} := \{k \in [n] : y_k = 0 \text{ for all } y \in P_{i,j}\} \quad \text{and} \quad A'_{i,j,0} := \{k \in [n] : y_k = 0 \text{ for all } y \in P'_{i,j}\}$$

CLAIM 7.6. *If execution of  $ALG'$  on  $f$  is regular, then the outcome of  $ALG'$  satisfies Condition  $C_1$ .*

*Proof.* Let  $(Q; P; R; \rho)$  be the final outcome of  $ALG'$  on  $f$ . For each  $i, j$  with  $P_{i,j} \neq \emptyset$ , let

$$A_{i,1} := \{k \in [n] : y_k = 1 \text{ for all } y \in P_i\} \quad \text{and} \quad A_{i,j,0} := \{k \in [n] : y_k = 0 \text{ for all } y \in P_{i,j}\},$$

with  $A_{i,j,1}$  being defined similarly (using  $y_k = 1$  instead of 0).

Given that the execution of  $ALG'$  on  $f$  is regular, we have from the definition above that

$$3n/4 + 1 \geq |A_{i,1}| \geq (3n/4) - 100|P_i| \log n \quad \text{and} \quad n/4 \geq |A_{i,j,0}| \geq (n/4) - 100|P_{i,j}| \log n.$$

So  $A_{i,j,0}$  already satisfies (7.4) given that  $P_{i,j} \subseteq P_i$ . We focus on  $A_{i,j,1}$  below. For any  $x, y \in P_{i,j}$ ,

$$|\{k \in [n] : x_k = y_k = 0\}| \geq |A_{i,j,0}| \geq (n/4) - 100|P_{i,j}| \log n.$$

Given that  $|x|, |y| \in \{3n/4, 3n/4 + 1\}$ , we have

$$|\{k \in [n] : x_k = 1, y_k = 0\}| \leq 100|P_{i,j}| \log n.$$

As a result, we fix any  $x \in P_i$  and have

$$|A_{i,j,1}| \geq |x| - \sum_{y \in P_{i,j}} |\{k \in [n] : x_k = 1, y_k = 0\}| \geq 3n/4 - O(|P_{i,j}|^2 \log n).$$

Combining this with  $3n/4 + 1 \geq |A_{i,j,1}| \geq |A_{i,1}| \geq (3n/4) - 100|P_i| \log n$  finishes the proof.  $\square$

CLAIM 7.7. *The execution of  $ALG'$  on  $\mathbf{f} \sim \mathcal{D}_{\text{yes}}$  is regular with probability at least  $1 - o(1)$ .*

*Proof.* Fix any outcome  $(Q; P; R; \rho)$  that is revealed to  $ALG'$  after  $k$  queries,  $k \leq q$ , such that the execution so far is regular, and let  $x$  be the next query. We show below that, conditioning on  $\mathbf{f} \sim \mathcal{D}_{\text{yes}}$  reaching  $(Q; P; R; \rho)$ , the probability that the execution of  $ALG'$  is no longer regular after querying  $x$  is  $o(1/q)$ . The lemma then follows by applying a union bound on  $k$ .

To this end, notice that by definition, the execution of  $ALG'$  remains regular after querying  $x$  unless either (1) there exists an  $i \in [L]$  such that  $P_i \neq \emptyset$ ,  $x$  satisfies  $\mathbf{T}_i$  (and thus, could be added to  $P_i$  potentially) but

$$\Delta_i := \{k \in [n] : k \in A_{i,1} \text{ but } x_k = 0\}$$

has size at least  $100 \log n$ ; or (2) there exist  $i, j \in [L]$  such that  $P_{i,j} \neq \emptyset$ ,  $x$  violates  $\mathbf{C}_{i,j}$  but

$$\Delta_{i,j} := \{k \in [n] : k \in A_{i,j,0} \text{ but } x_k = 1\}$$

has size at least  $100 \log n$ . In the rest of the proof, we show that (1) happens with probability at most  $o(1/q^2)$ . The same upper bound for (2) follows from a symmetric argument. The lemma then follows by a union bound over the  $O(q)$  many nonempty  $P_i$  and  $P_{i,j}$ 's.

For (1) and any fixed  $i$  with  $P_i \neq \emptyset$ , assuming that  $|\Delta_i| \geq 100 \log n$ , the conditional probability we are interested in can be written as  $|U|/|V|$ , where  $V$  is the set of  $T : [n] \rightarrow [n]$  such that

$$T(k) \in A_{i,1} \text{ for all } k \in [n] \text{ and each } z \in R_i \text{ has } z_{T(k)} = 0 \text{ for some } k \in [n]$$

and  $V$  is the set of  $T : [n] \rightarrow [n]$  such that

$$T(k) \in A_{i,1} \setminus \Delta_i \text{ for all } k \in [n] \text{ and each } z \in R_i \text{ has } z_{T(k)} = 0 \text{ for some } k \in [n].$$

Let  $\ell = \log n$ . First we write  $U'$  to denote the following subset of  $U$ :  $T \in U'$  if  $T \in U$  and the number of  $k \in [n]$  with  $T(k) \in \Delta_i$  is  $\ell$ . It suffices to show that  $|V|/|U'| = o(1/q^2)$  given that  $U'$  is a subset of  $U$ . Next we define the following bipartite graph  $G$  between  $U'$  and  $V$ :  $T' \in U'$  and  $V \in V$  have an edge if and only if  $T'(k) = T(k)$  for all  $k \in [n]$  with  $T'(k) \notin \Delta_i$ . From the construction, it is clear each  $T' \in U'$  has degree  $|A_{i,1} \setminus \Delta_i|^\ell$ .

To lowerbound the degree of a  $T \in V$ , note that one only needs at most  $q$  many variables of  $T$  to kill all strings in  $R_i$ . As a result, the degree of each  $T \in V$  is at least

$$\binom{n-q}{\ell} |\Delta_i|^\ell$$

By counting the number of edges in  $G$  in two different ways and using  $|A_{i,1}| \leq n$  trivially, we have

$$\frac{|U'|}{|V|} \geq \binom{n-q}{\ell} \cdot \left( \frac{|\Delta_i|}{|A_{i,1} \setminus \Delta_i|} \right)^\ell \geq \left( \frac{n-q}{\ell} \cdot \frac{100\ell}{3/4n+1} \right)^\ell \gg q^2.$$

This finishes the proof of the lemma.  $\square$

## Acknowledgements

X.C. is supported by NSF grants CCF-2106429 and CCF-2107187. A.D. is supported by NSF grants CCF-1910534 and CCF-2045128. Y.H. is supported by NSF grants CCF-2106429, CCF-2211238, CCF-2312242, CCF-2238221, a Google CyberNYC grant, an Amazon Research Award, and a grant from the Columbia Center of Artificial Intelligence Technology. Y.L. is supported by NSF grants CCF-2106429, CCF-2107187, and CCF-2211238. S.N. is supported by NSF grants CCF-2106429, CCF-2211238, CCF-1763970, and CCF-2107187. R.A.S. is supported by NSF grants CCF-2106429 and CCF-2211238. T.Y. is supported by NSF grants CCF-2106429, CCF-2107187, CCF-2211238, CCF-2212135, CCF-2238221, CCF-2312242, CAREER award CCF-2144219, a Google CyberNYC grant, an Amazon Research Award, AFOSR award FA9550-21-1-0040, and the Sloan Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors.

## References

- [BB16] A. Belovs and E. Blais. A polynomial lower bound for testing monotonicity. In *Proceedings of the 48th ACM Symposium on Theory of Computing*, 2016.
- [BB20] Eric Blais and Abhinav Bommireddi. On testing and robust characterizations of convexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 18:1–18:15, 2020.
- [BBBY12] Maria-Florina Balcan, Eric Blais, Avrim Blum, and Liu Yang. Active Property Testing. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 21–30, 2012.
- [BCO<sup>+</sup>15] E. Blais, C. Canonne, I. Oliveira, R. Servedio, and L.-Y. Tan. Learning circuits with few negations. In *Proceedings of the 15th Int. Workshop on Randomization and Computation (RANDOM)*, pages 512–527, 2015.
- [Bla09] Eric Blais. Testing juntas nearly optimally. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 2009.
- [BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993. Earlier version in STOC'90.
- [Bsh19] Nader H. Bshouty. Almost optimal distribution-free junta testing. In *34th Computational Complexity Conference, CCC*, pages 2:1–2:13, 2019.
- [Bsh20] Nader H. Bshouty. Almost optimal testers for concise representations. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 5:1–5:20, 2020.
- [CDS20] Clément L. Canonne, Anindya De, and Rocco A. Servedio. Learning from satisfying assignments under continuous distributions. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 82–101. SIAM, 2020.
- [CDST15] Xi Chen, Anindya De, Rocco A. Servedio, and Li-Yang Tan. Boolean Function Monotonicity Testing Requires (Almost)  $n^{1/2}$  Non-adaptive Queries. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 519–528, 2015.

- [CGM11] Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Efficient sample extractors for juntas with applications. In *Automata, Languages and Programming - 38th International Colloquium, ICALP*, pages 545–556, 2011.
- [CLS<sup>+</sup>18] Xi Chen, Zhengyang Liu, Rocco A. Servedio, Ying Sheng, and Jinyu Xie. Distribution-free junta testing. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2018.
- [CS13a] Deeparnab Chakrabarty and C. Seshadhri. A  $o(n)$  monotonicity tester for boolean functions over the hypercube. In *Proceedings of the 45th ACM Symposium on Theory of Computing*, pages 411–418, 2013.
- [CS13b] Deeparnab Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Symposium on Theory of Computing Conference, STOC*, pages 419–428, 2013.
- [CST14] Xi Chen, Rocco A. Servedio, and Li-Yang Tan. New algorithms and lower bounds for testing monotonicity. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science*, pages 286–295, 2014.
- [CWX17] Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond Talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 523–536, 2017.
- [DDS15] A. De, I. Diakonikolas, and R. A. Servedio. Learning from satisfying assignments. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 478–497, 2015.
- [DLM<sup>+</sup>07] I. Diakonikolas, H. Lee, K. Matulef, K. Onak, R. Rubinfeld, R. Servedio, and A. Wan. Testing for concise representations. In *Proc. 48th Ann. Symposium on Computer Science (FOCS)*, pages 549–558, 2007.
- [Fis01] E. Fischer. The art of uninformed decisions: A primer to property testing. *Computational Complexity Column of The Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, 2001.
- [FLN<sup>+</sup>02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proc. 34th Annual ACM Symposium on the Theory of Computing*, pages 474–483, 2002.
- [GGL<sup>+</sup>00] Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.
- [GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998.
- [Gol10] O. Goldreich, editor. *Property Testing: Current Research and Surveys*. Springer, 2010. LNCS 6390.
- [GR02] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.
- [HK07] S. Halevy and E. Kushilevitz. Distribution-Free Property Testing. *SIAM J. Comput.*, 37(4):1107–1138, 2007.
- [KMS03] Marcos A. Kiwi, Frédéric Magniez, and Miklos Santha. Approximate testing with error relative to input size. *J. Comput. Syst. Sci.*, 66(2):371–392, 2003.
- [KMS18] Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric-type theorems. *SIAM J. Comput.*, 47(6):2238–2276, 2018.
- [KR00] M. Kearns and D. Ron. Testing problems with sub-learning sample complexity. *Journal of Computer and System Sciences*, 61:428–456, 2000.
- [Mag00] Frédéric Magniez. Multi-linearity self-testing with relative error. In *STACS 2000*, pages 302–313, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- [MORS10] K. Matulef, R. O’Donnell, R. Rubinfeld, and R. Servedio. Testing halfspaces. *SIAM J. on Comput.*, 39(5):2004–2047, 2010.
- [O’D14] R. O’Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.
- [PR02] Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Struct. Algorithms*, 20(2):165–183, 2002.
- [PRR06] Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.
- [Ron08] D. Ron. Property Testing: A Learning Theory Perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.
- [RT14] Dana Ron and Gilad Tsur. Testing properties of sparse images. *ACM Trans. Algorithms*, 10(4):17:1–17:52, 2014.
- [Rub06] R. Rubinfeld. Sublinear time algorithms. Proceedings of the International Congress of Mathematicians (ICM), 2006.
- [RV04] Luis Rademacher and Santosh Vempala. Testing geometric convexity. In *Foundations of Software Technology and Theoretical Computer Science*, pages 469–480, 2004.

## A Separating standard and relative-error testing

We give a simple example of a property which provides a strong separation between the standard property testing model and the relative-error testing model. The property is as follows: for  $n$  a multiple of three, let  $\mathcal{C}$  be the class of all functions  $g : \{0, 1\}^n \rightarrow \{0, 1\}$  for which  $|g^{-1}(1)|$  is an integer multiple of  $2^{2n/3}$ .

For any  $\varepsilon \geq 2^{-n/3}$ , it is clear that every  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  has  $\text{ham-dist}(f, g) \leq \varepsilon$  for some function in  $\mathcal{C}$ , so

for such  $\varepsilon$  there is a trivial 0-query standard-model  $\varepsilon$ -testing algorithm (which always outputs “accept”). On the other hand, fix any  $\varepsilon < 1$  and consider the following two distributions over functions from  $\{0, 1\}^n$  to  $\{0, 1\}$ :

- $\mathcal{D}_{\text{yes}}$ : A draw of  $\mathbf{f} \sim \mathcal{D}_{\text{yes}}$  is obtained by choosing a uniformly random function from the set of all functions that output 1 on exactly  $2^{2n/3}$  inputs in  $\{0, 1\}^n$ .
- $\mathcal{D}_{\text{no}}$ : A draw of  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$  is obtained by choosing a uniformly random function from the set of all functions that output 1 on exactly  $\frac{1}{2}2^{2n/3}$  inputs in  $\{0, 1\}^n$ .

It is clear that every function in the support of  $\mathcal{D}_{\text{yes}}$  belongs to  $\mathcal{C}$ , whereas every function in the support of  $\mathcal{D}_{\text{no}}$  has  $\text{rel-dist}(\mathbf{f}, \mathcal{C}) = 1 > \varepsilon$ . We claim that any relative-error  $\varepsilon$ -testing algorithm for  $\mathcal{C}$  must make  $\Omega(2^{n/3})$  black-box queries or samples. To see this, first observe that any algorithm making  $o(2^{n/3})$  black-box queries has only an  $o(1)$  chance of every querying an input  $x$  such that  $\mathbf{f}(x) = 1$ , whether  $\mathbf{f}$  is drawn from  $\mathcal{D}_{\text{yes}}$  or from  $\mathcal{D}_{\text{no}}$ , and thus the two distributions are indistinguishable from  $o(2^{n/3})$  queries. But it is also the case that any algorithm making  $m = o(2^{n/3})$  calls to  $\text{Samp}(\mathbf{f})$  will with probability  $1 - o(1)$  receive an identically distributed sample in both cases ( $\mathbf{f} \sim \mathcal{D}_{\text{yes}}$  versus  $\mathbf{f} \sim \mathcal{D}_{\text{no}}$ ), since in both cases the sample will consist of  $m$  independent uniform distinct elements of  $\{0, 1\}^n$ .

## B Both oracles are needed for relative-error monotonicity testing

**A sampling oracle alone doesn’t suffice:** Let  $A$  be any algorithm which uses only the sampling oracle to do relative-error monotonicity testing and makes  $q$  calls to the sampling oracle, where  $q$  is at most (roughly)  $2^{n/2}$ . Consider the execution of  $A$  on a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  which is either (1) the constant-1 function, or (2) a uniform random Boolean function. The distribution of the sequence of samples received by  $A$  will be almost identical in these two cases, since in both cases with high probability this will be a uniform random sequence of  $q$  distinct elements of  $\{0, 1\}^n$ . Since in case (1) the function  $f$  is monotone and in case (2) with overwhelmingly high probability  $f$  has relative-distance at least 0.49 from every monotone function, it follows that such an algorithm  $A$  cannot be a successful relative-error monotonicity tester.

**A black-box oracle alone doesn’t suffice:** Let  $A$  be any algorithm which uses only a black-box oracle to do relative-error monotonicity testing and makes  $q$  calls to the black-box oracle, where  $q$  is at most  $2^{n/2}$ . Consider the execution of  $A$  on a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  which is either

1.  $f(x) = \mathbf{1}[x_1 + \cdots + x_n \geq 9n/10]$ , or
2. Let  $\mathbf{S}$  be a uniform random subset of  $2^{0.47n}$  of the  $\binom{n}{n/2} = \Theta(2^n/\sqrt{n})$  many strings in  $\{0, 1\}^n$  that have exactly  $n/2$  ones. The function  $f$  outputs 1 on input  $x$  if either  $x_1 + \cdots + x_n \geq 9n/10$  or if  $x \in \mathbf{S}$ .

In case (2), a simple argument shows that any  $2^{n/2}$ -query algorithm such as  $A$  will only have an  $o(1)$  probability of querying an input (an element of  $\mathbf{S}$ ) on which case (1) and case (2) disagree, so with  $1 - o(1)$  probability the sequence of responses to queries that  $A$  receives will be identical in these two cases. Since in case (1) the function  $f$  is monotone and in case (2)  $f$  has relative-distance at least 0.49 from every monotone function, such an algorithm  $A$  cannot be a successful relative-error monotonicity tester.