COMBINATORICA
Bolyai Society – Springer-Verlag

COMBINATORICA **20** (3) (2000) 301–337

TESTING MONOTONICITY*

ODED GOLDREICH†, SHAFI GOLDWASSER‡, ERIC LEHMAN, DANA RON§, ALEX SAMORODNITSKY

Received March 29, 1999

We present a (randomized) test for monotonicity of Boolean functions. Namely, given the ability to query an unknown function $f:\{0,1\}^n \to \{0,1\}$ at arguments of its choice, the test always accepts a monotone f, and rejects f with high probability if it is ϵ-far from being monotone (i.e., every monotone function differs from f on more than an ϵ fraction of the domain). The complexity of the test is $O(n/\epsilon)$.

The analysis of our algorithm relates two natural combinatorial quantities that can be measured with respect to a Boolean function; one being global to the function and the other being local to it. A key ingredient is the use of a *switching* (or *sorting*) operator on functions.

1. Introduction

In this work we address the problem of *testing whether a given Boolean function is monotone*. A function $f:\{0,1\}^n \to \{0,1\}$ is said to be *monotone* if $f(x) \le f(y)$ for every $x \prec y$, where $\prec$ denotes the natural partial order among strings (i.e., $x_1 \cdots x_n \prec y_1 \cdots y_n$ if $x_i \le y_i$ for every i and $x_i < y_i$ for some i). The testing algorithm can request the value of the function on

Mathematics Subject Classification (1991): 68Q25, 68R05, 68Q05

* A preliminary (and weaker) version of this work appeared in [25]

† Work done while visiting LCS, MIT.

‡ Supported in part by DARPA grant DABT63-96-C-0018 an in part by a Guastella fellowship.

§ This work was done while visiting LCS, MIT, and was supported by an ONR Science Scholar Fellowship at the Bunting Institute.

arguments of its choice, and is required to distinguish monotone functions from functions that are far from being monotone.

More precisely, the testing algorithm is given a *distance* parameter $\epsilon > 0$, and oracle access to an unknown function f mapping $\{0,1\}^n$ to $\{0,1\}$. If f is a monotone then the algorithm should accept it with probability at least $2/3$, and if f is at distance greater than ϵ from any monotone function then the algorithm should reject it with probability at least $2/3$. Distance between functions is measured in terms of the fraction of the domain on which the functions differ. The complexity measures we focus on are the *query complexity* and the *running time* of the testing algorithm.

We present a randomized algorithm for testing the monotonicity property whose query complexity and running time are linear in n and $1/\epsilon$. The algorithm performs a simple local test: It verifies whether monotonicity is maintained for randomly chosen pairs of strings that differ exactly on a single bit. In our analysis we relate this local measure to the global measure we are interested in — the minimum distance of the function to any monotone function.

1.1. Perspective

Property Testing, as explicitly defined by Rubinfeld and Sudan [36] and extended in [26], is best known by the special case of *low degree testing*[1] (see for example [17, 24, 36, 35, 7]), which plays a central role in the construction of probabilistically checkable proofs (PCP) [9, 8, 22, 6, 5, 35, 7]. The recognition that property testing is a general notion has been implicit in the context of PCP: It is understood that low degree tests as used in this context are actually codeword tests (in this case of BCH codes), and that such tests can be defined and performed also for other error-correcting codes such as the Hadamard Code [5, 13, 14, 11, 12, 33, 37], and the "Long Code" [12, 29, 30, 37].

For as much as error-correcting codes emerge naturally in the context of PCP, they do not seem to provide a *natural* representation of objects whose properties we may wish to investigate. That is, one can certainly encode any given object by an error-correcting code — resulting in a (legitimate yet) probably unnatural representation of the object — and then test properties of the encoded object. However, this can hardly be considered as a "natural test" of a "natural phenomena". For example, one may indeed represent a graph by applying an error correcting code to its adjacency matrix (or to its

[1] That is, testing whether a function (over some finite field) is a polynomial of some bounded degree d, or whether it differs significantly from any such polynomial.

incidence list), but the resulting string is not the "natural representation" of the graph.

The study of Property Testing as applied to natural representation of non-algebraic objects was initiated in [26]. In particular, Property Testing as applied to *graphs* has been studied in [26–28,2,3,34,15], where graphs are either represented by their adjacency matrix (most adequate for dense graphs), or by their incidence lists (adequate for sparse graphs).

In this work we consider property testing as applied to the most generic (i.e., least structured) object – an arbitrary Boolean function. In this case the choice of representation is "forced" upon us.

1.2. Monotonicity

In interpreting monotonicity it is useful to view Boolean functions over $\{0,1\}^n$ as subsets of $\{0,1\}^n$, called *concepts*. This view is the one usually taken in the PAC Learning literature. Each position in $\{1,\dots,n\}$ corresponds to a certain *attribute*, and a string $x = x_1 \cdots x_n \in \{0,1\}^n$ represents an instance where $x_i = 1$ if and only if the instance x has the i^{th} attribute. Thus, a concept (subset of instances) is monotone if the presence of additional attributes maintains membership of instances in the concept (i.e., if instance x is in the concept C then any instance resulting from x by adding some attributes is also in C).

The class of monotone concepts is quite general and rich. On the other hand, monotonicity suggests a certain aspect of simplicity. Namely, each attribute has a uni-directional effect on the value of the function. Thus, knowing that a concept is monotone may be useful in various applications. In fact, this form of simplicity is exploited by Angluin's learning algorithm for monotone concepts [4], which uses membership queries and has complexity that is linear in the number of terms in the DNF representation of the target concept.

We note that an efficient tester for monotonicity is useful as a preliminary stage before employing Angluin's algorithm. As is usually the case, Angluin's algorithm relies on the premise that the unknown target concept is in fact monotone. It is possible to simply apply the learning algorithm without knowing whether the premise holds, and hope that either the algorithm will succeed nonetheless in finding a good hypothesis or detect that the target is not monotone. However, due to the dependence of the complexity of Angluin's algorithm on the number of terms of the target concept's DNF representation, it may be much more efficient to first test whether the function is at all monotone (or close to it).

1.3. The natural monotonicity test

In this paper we show that a tester for monotonicity is obtained by repeating the following $O(n/\epsilon)$ times: Uniformly select a pair of strings at Hamming distance 1 and check if monotonicity is satisfied with respect to the value of f on these two strings. That is,

Algorithm 1. On input n, ϵ and oracle access to $f:\{0,1\}^n \to \{0,1\}$, repeat the following steps up to n/ϵ times

1. Uniformly select $x = x_1 \cdots x_n \in \{0,1\}^n$ and $i \in \{1, \ldots, n\}$.
2. Obtain the values of $f(x)$ and $f(y)$, where y results from x by flipping the i^{th} bit (that is, $y = x_1 \cdots x_{i-1}\, \bar{x}_i\, x_{i+1} \cdots x_n$).
3. If $x, y, f(x), f(y)$ demonstrate that f is not monotone then *reject*.
   That is, if either $(x \prec y) \wedge (f(x) > f(y))$ or $(y \prec x) \wedge (f(y) > f(x))$ then *reject*.

If all iterations are completed without rejecting then *accept*.

Theorem 1. *Algorithm 1 is a testing algorithm for monotonicity. Furthermore, if the function is monotone then Algorithm 1 always accepts.*

Theorem 1 asserts that a (random) *local check* (i.e., Step 3 above) can establish the existence of a *global property* (i.e., the distance of f to the set of monotone functions). Actually, Theorem 1 is proven by relating two quantities referring to the above: Given $f : \{0,1\}^n \to \{0,1\}$, we denote by $\delta_{\mathrm{M}}(f)$ the fraction of pairs of n-bit strings, differing on one bit that violate the monotonicity condition (as stated in Step 3). We then define $\epsilon_{\mathrm{M}}(f)$ to be the distance of f from the set of monotone functions (i.e., the minimum over all monotone functions g of $|\{x : f(x) \neq g(x)\}|/2^n$). Observing that Algorithm 1 always accepts a monotone function, Theorem 1 follows from Theorem 2, stated below.

Theorem 2. *For any $f:\{0,1\}^n \to \{0,1\}$,*

$$\delta_{\mathrm{M}}(f) \geq \frac{\epsilon_{\mathrm{M}}(f)}{n}.$$

On the other hand,

Proposition 3. *For every function $f:\{0,1\}^n \to \{0,1\}$, $\epsilon_{\mathrm{M}}(f) \geq \delta_{\mathrm{M}}(f)/2$.*

Thus, for every function f

$$\frac{\epsilon_{\mathrm{M}}(f)}{n} \leq \delta_{\mathrm{M}}(f) \leq 2 \cdot \epsilon_{\mathrm{M}}(f) \tag{1}$$

A natural question that arises is that of the exact relation between $\delta_{\mathrm{M}}(\cdot)$ and $\epsilon_{\mathrm{M}}(\cdot)$. We observe that this relation is not simple; that is, it does not depend only on the values of $\delta_{\mathrm{M}}(\cdot)$ and $\epsilon_{\mathrm{M}}(\cdot)$. Moreover, we show that both the lower and the upper bound of Equation (1) may be attained (up to a constant factor).

Proposition 4. *For every $c<1$, for any sufficiently large n, and for any α such that $2^{-c\cdot n} \le \alpha \le \frac{1}{2}$:*

1. *There exists a function $f:\{0,1\}^n \to \{0,1\}$ such that $\alpha \le \epsilon_{\mathrm{M}}(f) \le 2\alpha$ and*
$$\delta_{\mathrm{M}}(f) = \frac{2}{n} \cdot \epsilon_{\mathrm{M}}(f).$$
2. *There exists a function $f:\{0,1\}^n \to \{0,1\}$ such that $(1-o(1))\cdot\alpha \le \epsilon_{\mathrm{M}}(f) \le 2\alpha$ and*
$$\delta_{\mathrm{M}}(f) = (1 \pm o(1)) \cdot (1-c) \cdot \epsilon_{\mathrm{M}}(f).$$

Perspective. Analogous quantities capturing local and global properties of functions were analyzed in the context of *linearity testing*. For a function $f:\{0,1\}^n \to \{0,1\}$ (as above), one may define $\epsilon_{\mathrm{LIN}}(f)$ to be its distance from the set of linear functions and $\delta_{\mathrm{LIN}}(f)$ to be the fraction of pairs, $(x,y) \in \{0,1\}^n \times \{0,1\}^n$ for which $f(x)+f(y) \neq f(x \oplus y)$. A sequence of works [17, 13, 14, 11] has demonstrated a fairly complex behavior of the relation between $\delta_{\mathrm{LIN}}(\cdot)$ and $\epsilon_{\mathrm{LIN}}(\cdot)$. The interested reader is referred to [11].

Previous Bound on $\delta_{\mathrm{M}}(f)$. This paper is the journal version of [25]. In [25], a weaker version of Theorem 2 was proved. In particular it was shown that $\delta_{\mathrm{M}}(f) = \Omega\left(\frac{\epsilon_{\mathrm{M}}(f)}{n^2 \log(1/\epsilon_{\mathrm{M}}(f))}\right)$, thus yielding a testing algorithm whose complexity grows quadratically with n instead of linearly (as done here). Furthermore, the proof was more involved and the techniques did not lend themselves to obtain the results obtained subsequently (and presented in this paper) for testing monotonicity over domain alphabets other than $\{0,1\}$.

1.4. Monotonicity testing based on random examples

Algorithm 1 makes essential use of queries. We show that this is no coincidence – any monotonicity tester that utilizes only uniformly and *independently* chosen random examples, must have much higher complexity.

Theorem 5. *For any $\epsilon = O(n^{-3/2})$, any tester for monotonicity that only utilizes random examples must use at least $\Omega(\sqrt{2^n/\epsilon})$ such examples.*

Interestingly, this lower bound is tight (up to a constant factor).

Theorem 6. *There exists a tester for monotonicity that only utilizes random examples and uses at most $O(\sqrt{2^n/\epsilon})$ examples, provided*[2] $\epsilon > n^2 \cdot 2^{-n}$. *Furthermore, the algorithm runs in time* $\mathrm{poly}(n) \cdot \sqrt{2^n/\epsilon}$.

We note that the above tester is significantly faster than any learning algorithm for the class of all monotone concepts when the allowed error is $O(1/\sqrt{n})$: Learning (under the uniform distribution) requires $\Omega(2^n/\sqrt{n})$ examples (and at least that many queries) [31].[3]

1.5. Extensions

1.5.1. Other domain alphabets. Let Σ be a finite alphabet, and $<_\Sigma$ a (total) order on Σ. Then we can extend the notion of monotonicity to Boolean functions over Σ^n, in the obvious manner: Namely, a function $f : \Sigma^n \to \{0,1\}$ is said to be *monotone* if $f(x) \le f(y)$ for every $x \prec_\Sigma y$, where $x_1 \cdots x_n \prec_\Sigma y_1 \cdots y_n$ if $x_i \le_\Sigma y_i$ for every i and $x_i <_\Sigma y_i$ for some i.

A straightforward generalization of our algorithm yields a testing algorithm for monotonicity of functions over Σ^n with complexity $O\left(|\Sigma| \cdot \frac{n}{\epsilon}\right)$. By modifying the algorithm we can obtain a dependence on $|\Sigma|$ that is only logarithmic instead of linear. By an alternative modification we can remove the dependence on $|\Sigma|$ completely at the cost of increasing the dependence on n/ϵ from linear to quadratic.

1.5.2. Other ranges. We may further extend the notion of monotonicity to finite ranges other than $\{0,1\}$: Let Ξ be a finite set and $<_\Xi$ a (total) order on Ξ. We say that a function $: \Sigma^n \to \Xi$ is *monotone* if $f(x) \le_\Xi f(y)$ for every $x \prec_\Sigma y$. We show that every algorithm for testing monotonicity of Boolean function that works by observing pairs of strings selected according to some fixed distribution (as our algorithms do), can be transformed to testing monotonicity of functions over any finite range Ξ. The increase in

[2] For $\epsilon \le n^2 \cdot 2^{-n}$, an algorithm that obtains $O(n \cdot 2^n) = \mathrm{poly}(n) \cdot \sqrt{2^n/\epsilon}$ examples, can fully recover the function, and so easily determine whether it is monotone.

[3] This lower bound on the number of examples (or queries) can be derived by considering the following subclass of monotone concepts. Each concept in the class contains all instances having $\lfloor n/2 \rfloor + 1$ or more 1's, no instances having $\lfloor n/2 \rfloor - 1$ or less 1's, and some subset of the instances having exactly $\lfloor n/2 \rfloor$ 1's. In contrast, "weak learning" [32] is possible in polynomial time. Specifically, the class of monotone concepts can be learned in polynomial time with error at most $1/2 - \Omega(1/\sqrt{n})$ [16] (though no polynomial-time learning algorithm can achieve an error of $1/2 - \omega(\log(n)/\sqrt{n})$) [16]).

the complexity of the algorithm is by a multiplicative factor of $|\Xi|$. Recently, Doddis, Lehman and Raskhodnikova have devised a transformation whose dependency on the size of the range is only logarithmic [20].

1.5.3. Testing unateness. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be *unate* if for every $i \in \{1,\dots,n\}$ exactly one of the following holds: whenever the i^{th} bit is flipped from 0 to 1 then the value of f does not decrease; *or* whenever the i^{th} bit is flipped from 1 to 0 then the value of f does not decrease. Thus, unateness is a more general notion than monotonicity. We show that our algorithm for testing monotonicity of Boolean functions over $\{0,1\}^n$ can be extended to test whether a function is unate or far from any unate function at an additional cost of a (multiplicative) factor of $\sqrt{n}$.

1.6. Techniques

Our main results are proved using *shifting* of Boolean functions (associated with subsets of $\{0,1\}^n$). Various shifting techniques play an important role in extremal set theory (cf., [23] as well as [1, 19]).

Shifting a Boolean function means modifying the set of inputs on which the value of the function is 1. The modification is chosen accordingly to the desired application. A typical application is for showing that a function has a certain property. This is done by shifting the function so that the resulting function is simpler to analyze, whereas shifting *does not introduce* the property in question.

Our applications are different. We shift the function to make it monotone, while using a "charging" operator to account for the number of changes made by the shifting process. This "charge" is on one hand related to the distance of the function from being monotone, and on the other hand related to the local check conducted by our testing algorithm.

Actually we will be using several names for the same procedure – *sorting* and *switching* will also make an appearance.

1.7. Related work

The "spot-checker for sorting" presented in [21, Sec. 2.1] implies a tester for monotonicity with respect to functions from any fully ordered domain to any fully ordered range, having query and time complexities that are logarithmic in the size of the domain. We note that this problem corresponds to the special case of $n = 1$ of the extension discussed in Subsection 1.5 (to general domains and ranges).

1.8. An open problem

Our algorithm (even for the case $f:\{0,1\}^n \to \{0,1\}$), has a linear dependence on the dimension of the input, n. As shown in Proposition 4, this dependence on n is unavoidable in the case of our algorithm. However, it is an interesting open problem whether other algorithms may have significantly lower dependence on n.

Organization

Theorem 2 is proved in Section 3. Propositions 3 and 4 are proved in Section 4. The extension to domains alphabets and ranges other than $\{0,1\}$, is presented in Section 5, and the extension to testing Unateness is described in Section 6. Theorems 5 and 6 are proved in Section 7.

2. Preliminaries

For any pair of functions $f,g:\{0,1\}^n \to \{0,1\}$, we define the *distance* between f and g, denoted $\mathrm{dist}(f,g)$, to be the fraction of instances $x \in \{0,1\}^n$ on which $f(x) \neq g(x)$. In other words, $\mathrm{dist}(f,g)$ is the probability over a uniformly chosen x that f and g differ on x. Thus, $\epsilon_{\mathrm{M}}(f)$ as defined in the introduction is the minimum, taken over all monotone functions g of $\mathrm{dist}(f,g)$.

A general formulation of Property Testing was suggested in [26], but here we consider a special case formulated previously in [36].

Definition 1. *(property tester)*: Let $\mathrm{P} = \cup_{\mathrm{n} \geq 1} \mathrm{P_n}$ be a subset *(or a property)* of Boolean functions, so that P_n is a subset of the functions mapping $\{0,1\}^n$ to $\{0,1\}$. A *(property) tester for* P is a probabilistic oracle machine[4], M, which given n, a *distance parameter* $\epsilon > 0$ and oracle access to an arbitrary function $f:\{0,1\}^n \to \{0,1\}$ satisfies the following two conditions:

1. *The tester accepts f if it is in* P *:*
   If $f \in \mathrm{P}_n$ then $\mathrm{Prob}(M^f(n,\epsilon)=1) \geq \frac{2}{3}$.
2. *The tester rejects f if it is far from* P *:*
   If $\mathrm{dist}(f,g) > \epsilon$ for every $g \in \mathrm{P}_n$, then $\mathrm{Prob}(M^f(n,\epsilon)=1) \leq \frac{1}{3}$.

[4] Alternatively, one may consider a RAM model of computation, in which trivial manipulation of domain and range elements (e.g., reading/writing an element and comparing elements) is performed at unit cost.

Testing based on random examples [26]. In case the queries made by the tester are uniformly and *independently* distributed in $\{0,1\}^n$, we say that it *only uses examples*. Indeed, a more appealing way of looking as such a tester is as an ordinary algorithm (rather than an oracle machine), which is given as input a sequence $(x_1, f(x_1)), (x_2, f(x_2)), \ldots$ where the x_i's are uniformly and independently distributed in $\{0,1\}^n$.

3. Proof of Theorem 2

In this section we show how every function f can be transformed into a monotone function g. By definition of $\epsilon_{\mathrm{M}}(f)$, the number of modification performed in the transformation must be *at least* $\epsilon_{\mathrm{M}}(f) \cdot 2^n$. On the other hand, we shall be able to *upper bound* the number of modifications by $\delta_{\mathrm{M}}(f) \cdot n \cdot 2^n$, thus obtaining the bound on $\delta_{\mathrm{M}}(f)$ stated in Theorem 2.

Definition 2. For any $i \in \{1, \ldots, n\}$, we say that a function f is *monotone in dimension* i, if for every $\alpha \in \{0,1\}^{i-1}$ and $\beta \in \{0,1\}^{n-i}$, $f(\alpha 0 \beta) \leq f(\alpha 1 \beta)$. For a set of indices $\mathrm{T} \subseteq \{1, \ldots, n\}$, we say that f is *monotone in dimensions* T, if for every $i \in \mathrm{T}$, the function f is monotone in dimension i.

We next define a *switch* operator, S_i that transforms any function f to a function $S_i(f)$ that is monotone in dimension i.

Definition 3. For every $i \in \{1, \ldots, n\}$, the function $S_i(f) : \{0,1\}^n \to \{0,1\}$ is defined as follows: For every $\alpha \in \{0,1\}^{i-1}$ and every $\beta \in \{0,1\}^{n-i}$, if $f(\alpha 0 \beta) > f(\alpha 1 \beta)$ then $S_i(f)(\alpha 0 \beta) = f(\alpha 1 \beta)$, and $S_i(f)(\alpha 1 \beta) = f(\alpha 0 \beta)$. Otherwise, $S_i(f)$ is defined as equal to f on the strings $\alpha 0 \beta$ and $\alpha 1 \beta$.

Notation 4. Let

$$\mathrm{U} \stackrel{\text{def}}{=} \{(x,y) : \ x \text{ and } y \text{ differ on a single bit and } x \prec y\} \tag{2}$$

be the set of *neighboring pairs*, and let

$$\Delta(f) = \{(x,y) : (x,y) \in \mathrm{U} \text{ and } f(x) > f(y)\} \tag{3}$$

be the set of *violating (neighboring) pairs*. Hence, $|\mathrm{U}| = \frac{1}{2} \cdot 2^n \cdot n$, and by definition of $\delta_{\mathrm{M}}(f)$, we have $\delta_{\mathrm{M}}(f) = \frac{|\Delta(f)|}{|\mathrm{U}|}$. Let

$$D_i(f) \stackrel{\text{def}}{=} |\{x : S_i(f)(x) \neq f(x)\}| \tag{4}$$

so that $D_i(f)$ is twice the number of pairs in $\Delta(f)$ that differ on the i^{th} bit (and $\sum_{i=1}^{n} D_i(f) = 2 \cdot |\Delta(f)|$).

We show:

Lemma 7. *For every* $f:\{0,1\}^n \to \{0,1\}$ *and* $j \in [n]$, *we have:*

1. *If* f *is monotone in dimensions* $\mathrm{T} \subseteq [n]$ *then* $S_j(f)$ *is monotone in dimensions* $\mathrm{T} \cup \{j\}$;
2. *For every* $1 \le i \ne j \le n$, $D_j(S_i(f)) \le D_j(f)$.

We note that the first item in the lemma is actually a special case of the second item. However, for sake of the presentation we have chosen to state and prove it separately.

We prove the lemma momentarily. First we show how Theorem 2 follows. Let $g = S_n(S_{n-1}(\cdots(S_1(f))\cdots)$. By successive application of the first item of Lemma 7, the function g is monotone, and hence $\mathrm{dist}(f,g) \ge \epsilon_{\mathrm{M}}(f)$. By successive applications of the second item,

$$D_i(S_{i-1}(\cdots(S_1(f))\cdots) \le D_i(S_{i-2}(\cdots(S_1(f))\cdots) \le \cdots \le D_i(f) \tag{5}$$

and so

$$\mathrm{dist}(f,g) \le 2^{-n} \cdot \sum_{i=1}^{n} D_i(S_{i-1}(\cdots(S_1(f))\cdots) \le 2^{-n} \cdot \sum_{i=1}^{n} D_i(f). \tag{6}$$

Therefore,

$$\sum_{i=1}^{n} D_i(f) \ge \mathrm{dist}(f,g) \cdot 2^n \ge \epsilon_{\mathrm{M}}(f) \cdot 2^n \tag{7}$$

On the other hand, by definition of $D_i(f)$,

$$\sum_{i=1}^{n} D_i(f) = 2 \cdot |\Delta(f)| = 2 \cdot \delta_{\mathrm{M}}(f) \cdot |\mathrm{U}| = \delta_{\mathrm{M}}(f) \cdot 2^n \cdot n \tag{8}$$

where U and $\Delta(f)$ were defined in Equations (2) and (3), respectively. Theorem 2 follows by combining Equations (7) and (8).

Proof of Lemma 7. A key observation is that for every $i \ne j$, the effect of S_j on the monotonicity of f in dimension i (resp., the effect of S_i on the value of $D_j(\cdot)$) can be analyzed by considering separately each restriction of f at the other coordinates.

Item 1. Clearly, $S_j(f)$ is monotone in dimension j. We show that $S_j(f)$ is monotone in any dimension $i \in \mathrm{T}$. Fixing any $i \in \mathrm{T}$, and assuming without loss of generality, that $i < j$, we fix any $\alpha \in \{0,1\}^{i-1}$, $\beta \in \{0,1\}^{j-i-1}$ and $\gamma \in \{0,1\}^{n-j}$, and consider the function $f'(\sigma\tau) \stackrel{\mathrm{def}}{=} f(\alpha\sigma\beta\tau\gamma)$ where $\sigma,\tau \in \{0,1\}$. Clearly f' is monotone in dimension 1 and we need to show that so is $S_2(f')$. In other words, consider the 2-by-2 zero-one matrix whose (σ,τ)-entry is $f'(\sigma\tau)$. Our claim thus amounts to saying that if one sorts the rows of a 2-by-2 matrix whose columns are initially sorted then the columns remain sorted. This is easily verified by a simple case analysis. For a more general argument, concerning any $d\times d$ zero-one matrix, see the proof of Lemma 8.

Item 2. Fixing i,j,α,β,γ and defining f' as above, here we need to show that $D_2(S_1(f'))\leq D_2(f')$. Again, we consider the 2-by-2 zero-one matrix whose (σ,τ)-entry is $f'(\sigma\tau)$. The current claim amounts to saying that for any such matrix if we sort the columns then the number of unsorted rows cannot increase. (Recall that D_2 equals twice the number of unsorted rows.) The claim is easily verified by a simple case analysis. For a more general argument, concerning any $d\times 2$ zero-one matrix, see the proof of Lemma 8. (We note that the claim is false for $d\times d$ zero-one matrices, starting at $d\geq 4$ as well as for 2-by-2 matrices with non-binary entries – see Appendix.) ∎

4. Proofs of Propositions 3 and 4

Below we prove the propositions concerning the other relations between $\epsilon_{\mathrm{M}}(f)$ and $\delta_{\mathrm{M}}(f)$ that were stated in the introduction.

Proposition 3. *For every function* $f:\{0,1\}^n\to\{0,1\}$, $\epsilon_{\mathrm{M}}(f)\geq\delta_{\mathrm{M}}(f)/2$.

Proof. Let us fix f and consider the set $\Delta(f)$ of its violating pairs (as defined in Equation (3)). In order to make f monotone, we must modify the value of f on at least one string in each violating pair. Since each string belongs to at most n violating pairs, the number of strings whose value must be modified (i.e., $\epsilon_{\mathrm{M}}(f)\cdot 2^n$) is at least

$$\frac{|\Delta(f)|}{n}=\frac{\delta_{\mathrm{M}}(f)\cdot|\mathrm{U}|}{n}=\frac{\delta_{\mathrm{M}}(f)\cdot\left(\frac{1}{2}\cdot 2^n\cdot n\right)}{n}=\frac{\delta_{\mathrm{M}}(f)}{2}\cdot 2^n$$

(where U is as defined in Equation (2)), and the proposition follows. ∎

Comment. For each string z, if $f(z)=0$ then at most all pairs $(x,z)\in\mathrm{U}$ are violating, and if $f(z)=1$, then at most all pairs $(z,y)\in\mathrm{U}$ are violating. The number of former pairs equals the number of 1's in z and the number of latter pairs equals the number of 0's in z. Since all but a small fraction of strings have roughly $n/2$ 1's and $n/2$ 0's, the above bound can be improved to yield $\epsilon_{\mathrm{M}}(f)\geq(1-o(1))\cdot\delta_{\mathrm{M}}(f)$, provided $\delta_{\mathrm{M}}(f)\geq 2^{-cn}$ for every constant $c<1$.

Proposition 4. *For every $c<1$, for any sufficiently large n, and for any α such that $2^{-c\cdot n}\leq\alpha\leq\frac{1}{2}$:*

1. *There exists a function $f:\{0,1\}^n\to\{0,1\}$ such that $\alpha\leq\epsilon_{\mathrm{M}}(f)\leq 2\alpha$ and*
$$\delta_{\mathrm{M}}(f)=\frac{2}{n}\cdot\epsilon_{\mathrm{M}}(f).$$
2. *There exists a function $f:\{0,1\}^n\to\{0,1\}$ such that $(1-o(1))\cdot\alpha\leq\epsilon_{\mathrm{M}}(f)\leq 2\alpha$ and*
$$\delta_{\mathrm{M}}(f)=(1\pm o(1))\cdot(1-c)\cdot\epsilon_{\mathrm{M}}(f).$$

Proof. It will be convenient to view the Boolean Lattice as a directed layered graph G_n. Namely, each string in $\{0,1\}^n$ corresponds to a vertex in G_n. For every vertex $y=y_1\dots y_n$, and for every i such that $y_i=1$, there is an edge directed from y to $x=y_1\dots y_{i-1}0y_{i+1}\dots y_n$. Thus G_n is simply a directed version of the hypercube graph. We refer to all vertices corresponding to strings having exactly i 1's as belonging to the i^{th} *layer* of G_n, denoted L_i. By definition of the edges in the graph, there are only edges between consecutive layers. For any function $f:\{0,1\}^n\to\{0,1\}$, we say that an edge from y to x is *violating* with respect to f, if $f(x)>f(y)$ (which implies that $(x,y)\in\Delta(f)$). The fraction of violating edges (among all $\frac{1}{2}\cdot 2^n\cdot n$ edges), is by definition $\delta_{\mathrm{M}}(f)$.
We start by proving both items for the case where $\alpha=\frac{1}{2}-O(\frac{1}{\sqrt{n}})$.

Item 1. Let $f=g_n$ be defined on $\{0,1\}^n$ in the following way: $g_n(x)=1$ if $x_1=0$, and $g_n(x)=0$ if $x_1=1$ (thus g_n is the "dictatorship" function). By definition of g_n, for every $\beta\in\{0,1\}^{n-1}$, the edge $(1\beta,0\beta)$ is a violating edge with respect to g_n, and there are no other violating edges (since for every edge (y,x) such that $x_1=y_1$, we have $g_n(x)=g_n(y)$.) Since the number of violating edges is 2^{n-1} (as there is a single edge for each $\beta\in\{0,1\}^n$), and the total number of edges is $\frac{1}{2}\cdot 2^n\cdot n$, we have $\delta_{\mathrm{M}}(g_n)=\frac{2^{n-1}}{\frac{1}{2}\cdot 2^n\cdot n}=\frac{1}{n}$

On the other hand, we next show that $\epsilon_{\mathrm{M}}(g_n)=\frac{1}{2}$. Clearly, $\epsilon_{\mathrm{M}}\leq\frac{1}{2}$ as the all 0 function is monotone and at distance $\frac{1}{2}$ from g_n. It remains to show that we cannot do better. To this end, observe that the violating edges, of which there are 2^{n-1}, define a matching between $\mathrm{C}\stackrel{\text{def}}{=}\{y\in\{0,1\}^n:y_1=1\}$ and $\overline{\mathrm{C}}\stackrel{\text{def}}{=}\{x\in\{0,1\}^n:x_1=0\}$ (where for every $\beta\in\{0,1\}^{n-1}$, $y=1\beta$ is matched with $x=0\beta$). To make g_n monotone, we must modify the value of g_n on at least one vertex in each matched pair, and since these pairs are disjoint the claim follows.

Item 2. Let $f=h_n:\{0,1\}^n\to\{0,1\}$ be the (symmetric) function that has value 0 on all vertices belonging to layers L_i where i is even, and has value 1 on all vertices belonging to layers L_i where i is odd (i.e., h_n is the parity function). Since all edges going from even layers to odd layers are violating edges, $\delta_{\mathrm{M}}(h_n)=1/2$. We next show that $\epsilon_{\mathrm{M}}(h_n)\geq\frac{1}{2}-O(\frac{1}{\sqrt{n}})$ (where once again, $\epsilon_{\mathrm{M}}(h_n)\leq\frac{1}{2}$ since h_n is at distance at most $1/2$ either from the all-0 function or the all-1 function). Consider any pair of adjacent layers such that the top layer is labeled 0 (so that all edges between the two layers are violating edges). It can be shown (*cf.* [18, Chap. 2, Cor. 4]) using Hall's Theorem, that for any such pair of adjacent layers, there exists a perfect matching between the smallest among the two layers and a subset of the larger layer. The number of unmatched vertices is hence $\sum_{i=1}^{\lceil n/2\rceil}||\mathrm{L}_{2i}|-|\mathrm{L}_{2i-1}||+1$ (where $\mathrm{L}_{n+1}\stackrel{\text{def}}{=}\emptyset$, and the $+1$ is due to the all 0 string). This sum can be bounded by

$$
\begin{aligned}
&2+2\cdot\sum_{i=1}^{\lceil n/4\rceil}||\mathrm{L}_{2i}|-|\mathrm{L}_{2i-1}||\\
&\qquad=2+2\cdot\sum_{i=1}^{\lceil n/4\rceil}(|\mathrm{L}_{2i}|-|\mathrm{L}_{2i-1}|)\leq 2+2\cdot|\mathrm{L}_{\lceil n/2\rceil}|=O(2^n/\sqrt{n})
\end{aligned}
$$

Thus, we have at least $(1-o(1))\cdot 2^{n-1}$ disjoint violating edges. Since we must modify the value of at least one end-point of each violating edge, $\epsilon_{\mathrm{M}}(h_n)\in[0.5-o(1),0.5]$ and the claim follows.

To generalize the above two constructions for smaller α we do the following. For each value of α we consider a subset $\mathrm{S}\subset\{0,1\}^n$, such that all strings in S have a certain number of leading 0's, and the size of S is roughly $2\alpha\cdot 2^n$. Thus there is a 1-to-1 mapping between S and $\{0,1\}^{n'}$ for a certain n', and S induces a subgraph of G_n that is isomorphic to $\mathrm{G}_{n'}$. For both case we define f on S analogously to the way it was defined above on $\{0,1\}^n$, and let f be 1 everywhere else. We argue that the values of $\epsilon_{\mathrm{M}}(f)$ and $\delta_{\mathrm{M}}(f)$ are determined by the value of f on S, and adapt the bounds we obtained above. Details follow.

Item 1. Let $n'=n-\lfloor\log(1/(2\alpha))\rfloor$, and consider the set S of all strings whose first $n-n'$ bits are set to 0 (thus forming a sub-cube of the n-dimensional cube). The size of the set S is at least $2\alpha\cdot 2^n$ and at most $4\alpha\cdot 2^n$. Clearly the subgraph of G_n induced by vertices in S is isomorphic to $\mathrm{G}_{n'}$. For every $x=0^{n-n'}\gamma\in\mathrm{S}$ (where $\gamma\in\{0,1\}^{n'}$), we let $f(x)=g_{n'}(\gamma)$, (where $g_{n'}:\{0,1\}^{n'}\to\{0,1\}$ is as defined in the special case of Item 1 above), and for every $x\notin\mathrm{S}$, we let $f(x)=1$. Therefore, for every $x\in\mathrm{S}$ and $y\notin\mathrm{S}$, either $x\prec y$ or x and y are incomparable. This implies that the closest monotone functions differs from f only on S, and all violating edges (with respect to f) are between vertices in S. Therefore, $\epsilon_{\mathrm{M}}(f)=\frac{\epsilon_{\mathrm{M}}(h_{n'})\cdot 2^{n'}}{2^n}=\frac{|\mathrm{S}|/2}{2^n}$ (which ranges between α and 2α), and $\delta_{\mathrm{M}}(f)=\frac{\delta_{\mathrm{M}}(h_{n'})\cdot 2^{n'}n'/2}{2^n n/2}=\frac{|\mathrm{S}|/2}{2^n\cdot n/2}$. So $\delta_{\mathrm{M}}(f)=\frac{2\epsilon_{\mathrm{M}}(f)}{n}$, as desired.

Item 2. Here we let $n'=n-\lfloor\log(1/(2\alpha))\rfloor$, and define S as in Item 1. Thus, $2\alpha\cdot 2^n\le|\mathrm{S}|\le 4\alpha\cdot 2^n$. For every $x=0^{n-n'}\gamma\in\mathrm{S}$ (where $\gamma\in\{0,1\}^{n'}$), we let $f(x)=h_{n'}(\gamma)$, (where $h_{n'}:\{0,1\}^{n'}\to\{0,1\}$ is as defined in Item 2 above), and for every $x\notin\mathrm{S}$, we let $f(x)=1$. Therefore, $\epsilon_{\mathrm{M}}(f)=\frac{\epsilon_{\mathrm{M}}(g_{n'})\cdot 2^{n'}}{2^n}=\frac{(\frac{1}{2}\pm\frac{1}{\sqrt{n'}})\cdot|\mathrm{S}|}{2^n}$ (which is greater than $(1-o(1))\alpha\cdot 2^n$ and less than $2\alpha\cdot 2^n$), and $\delta_{\mathrm{M}}(f)=\frac{\delta_{\mathrm{M}}(g_{n'})\cdot 2^{n'}n'/2}{2^n n/2}=\frac{|\mathrm{S}|\cdot n'/4}{2^n\cdot n/2}$. We thus have $\delta_{\mathrm{M}}(f)=\frac{(1\pm o(1))\cdot n'}{n}\cdot\epsilon_{\mathrm{M}}(f)$. Since $n'>(1-c)\cdot n-3$, the claim follows. ∎

5. Other domain alphabets and ranges

As defined in the introduction, for finite sets Σ and Ξ and orders $<_\Sigma$ and $<_\Xi$ on Σ and Ξ, respectively, we say that a function $f:\Sigma^n\to\Xi$ is *monotone* if $f(x)\le_\Xi f(y)$ for every $x\prec_\Sigma y$, where $x_1\cdots x_n\prec_\Sigma y_1\cdots y_n$ if $x_i\le_\Sigma y_i$ for every i and $x_i<_\Sigma y_i$ for some i. In this subsection we discuss how our algorithm generalizes when Σ and Ξ are not necessarily $\{0,1\}$. We first consider the generalization to $|\Sigma|>2$ while maintaining $\Xi=\{0,1\}$, and later generalize to any Ξ.

5.1. General domain alphabets

Let $f:\Sigma^n\to\{0,1\}$, where $|\Sigma|=d$. Without loss of generality, let $\Sigma=\{1,\ldots,d\}$.[5] A straightforward generalization of Algorithm 1 uniformly selects a set of strings, and for each string x selected it uniformly select an index $j\in\{1,\ldots,n\}$, and queries the function f on x and y, where y is obtained from x by either incrementing or decrementing by one unit the value of x_j. However, as we shall see below, the number of strings that should be selected in order to obtain 2/3 success probability (using this algorithm), grows linearly with d. Instead, we show how a modification of the above algorithm, in which the distribution on the pairs (x,y) is different from the

[5] For sake of consistency with the binary case where $\Sigma=\{0,1\}$, we could let $\Sigma=\{0,\ldots,d-1\}$. However, this choice would make the presentation of our results in this section somewhat more cumbersome and hence we have chosen to use $\Sigma=\{1,\ldots,d\}$.

above, yields an improved performance. Both algorithms are special cases of the following algorithmic schema.

Algorithm 2. The algorithm utilizes a distribution $p:\Sigma\times\Sigma\to[0,1]$, and depends on a function t. Without loss of generality, $p(k,\ell)>0$ implies $k<\ell$. On input n,ϵ and oracle access to $f:\Sigma^n\to\{0,1\}$, repeat the following steps up to $t(n,\epsilon,|\Sigma|)$ times

1. Uniformly select $i\in\{1,\dots,n\}$, $\alpha\in\Sigma^{i-1}$, and $\beta\in\Sigma^{n-i}$.
2. Select (k,ℓ) according to the distribution p.
3. If $f(\alpha k\beta)>f(\alpha\ell\beta)$ (that is, a violation of monotonicity is detected), then *reject*.

If all iterations were completed without rejecting then *accept*.

The above algorithm clearly generalizes the algorithm suggested at the beginning of this section (where $t(n,\epsilon,d)=\Theta(n\cdot d/\epsilon)$ and the distribution p is uniform over $\{(k,k+1):1\le k<d\}$). However, as we show below, we can select the distribution p so that $t(n,\epsilon,d)=\Theta(\frac{n}{\epsilon}\cdot\log d)$ will do. Yet a third alternative (i.e., letting p be uniform over all pairs (k,ℓ) with $1\le k<\ell\le d$) allows to have $t(n,\epsilon,d)=O(n/\epsilon)^2$.

Clearly, Algorithm 2 always accepts a monotone function (regardless of the distribution p in use). Our analysis thus focuses on the case the function is not monotone.

5.1.1. Reducing the analysis to the case $n=1$.

We reduce the analysis of the performance of the above algorithm to its performance in the case $n=1$. The key ingredient in this reduction is a generalization of Lemma 7. As in the binary case, we describe operators by which any Boolean function over Σ^n can be transformed into a monotone function. In particular we generalize the switch operator (which is now a *sort* operator) to deal with the case $d>2$.

Definition 5. For every $i\in\{1,\dots,n\}$, the function $\mathrm{S}_i(f):\Sigma^n\to\{0,1\}$ is defined as follows: For every $\alpha\in\Sigma^{i-1}$ and every $\beta\in\Sigma^{n-i}$, we let $\mathrm{S}_i(f)(\alpha 1\beta),\dots,\mathrm{S}_i(f)(\alpha d\beta)$ be given the values of $f(\alpha 1\beta),\dots,f(\alpha d\beta)$, in *sorted* order.

Clearly, similarly to the binary case, for each i, the function $S_i(f)$ is monotone in dimension $\{i\}$, where the definition of being monotone in a set of dimensions is as in the binary case.[6] The definitions of U and $\Delta(f)\subseteq\mathrm{U}$

[6] That is, for $\mathrm{T}\subseteq\{1,\dots,n\}$, we say that the function $f:\Sigma^n\to\{0,1\}$ is *monotone in dimensions* T if for every $i\in\mathrm{T}$, every $\alpha\in\Sigma^{i-1},\beta\in\Sigma^{n-i}$, and every $k=1,\dots,d-1$, it holds that $f(\alpha k\beta)\le f(\alpha(k+1)\beta)$.

of the binary case (cf., Equations (2) and (3)) may be extended in several different ways. We use the following:

Notation 6. For every $i \in [n] \stackrel{\text{def}}{=} \{1,\ldots,n\}$ and every pair $(k,\ell) \in \Sigma^2$ so that $k < \ell$, we let

$$\mathrm{U}_{i,(k,\ell)} \stackrel{\text{def}}{=} \{(\alpha\, k\, \beta\,,\, \alpha\, \ell\, \beta) : \alpha \in \Sigma^{i-1}\,,\, \beta \in \Sigma^{n-i}\} \tag{9}$$

$$\Delta_{i,(k,\ell)}(f) \stackrel{\text{def}}{=} \{(x,y) \in \mathrm{U}_{i,(k,\ell)} : f(x) > f(y)\} \tag{10}$$

In the binary case (where here instead of $\Sigma = \{0,1\}$ we have $\Sigma = \{1,2\}$), $\mathrm{U} = \bigcup_{i=1}^{n} \mathrm{U}_{i,(1,2)}$ and $\Delta(f) = \bigcup_{i=1}^{n} \Delta_{i,(1,2)}(f)$. Furthermore, $D_i(f)$ as defined in the binary case, equals twice $|\Delta_{i,(1,2)}(f)|$.

Lemma 8. (Lemma 7 generalized): *For every $f : \Sigma^n \to \{0,1\}$ and $j \in [n]$, we have:*

1. *If f is monotone in dimensions $\mathrm{T} \subseteq [n]$ then $S_j(f)$ is monotone in dimensions $\mathrm{T} \cup \{j\}$;*
2. *For every $i \in [n] \setminus \{j\}$, and for every $1 \le k < \ell \le d$*

$$|\Delta_{j,(k,\ell)}(S_i(f))| \le |\Delta_{j,(k,\ell)}(f)|$$

Proof. As in the proof of Lemma 7, we may consider the function f restricted at all dimensions but the two in question. Again, the proofs of the two items boil down to corresponding claims about sorting matrices.

Item 1. Let i be some index in T, and assume without loss of generality that $i < j$. Again, we fix any $\alpha \in \Sigma^{i-1}$, $\beta \in \Sigma^{j-i-1}$ and $\gamma \in \Sigma^{n-j}$, and consider the function $f' : \Sigma^2 \to \{0,1\}$ defined by $f'(\sigma\tau) \stackrel{\text{def}}{=} f(\alpha\sigma\beta\tau\gamma)$. Again, f' is monotone in dimension 1 and we need to show that so is $S_2(f')$ (as it is obvious that $S_2(f')$ is monotone in dimension 2). Our claim thus amounts to saying that if one sorts the rows of a d-by-d matrix whose columns are sorted then the columns remain sorted (the matrix we consider has its (σ,τ)-entry equal to $f'(\sigma\tau)$).

Let M denote a (d-by-d zero-one) matrix in which each column is sorted. We observe that the number of 1's in the rows of M is monotonically non-decreasing (as each column contributes a unit to the 1-count of row k only if it contributes a unit to the 1-count of row $k+1$). That is, if we let o_k denote the number of 1's in the k^{th} row then $o_k \le o_{k+1}$ for $k = 1,\ldots,d-1$. Now suppose we sort each row of M resulting in a matrix M'. Then the k^{th} row of M' is $0^{d-o_k}1^{o_k}$, and it follows that the columns of M' remain sorted (as the $k+1^{\text{st}}$ row of M' is $0^{d-o_{k+1}}1^{o_{k+1}}$ and $o_k \le o_{k+1}$).

Item 2. Fixing i,j,α,β,γ and defining f' as above, here we need to show that $|\Delta_{2,(k,\ell)}(S_1(f'))| \le |\Delta_{2,(k,\ell)}(f')|$. The current claim amounts to saying that for any $d\times 2$ zero-one matrix if we sort the (two) columns then the number of unsorted rows cannot increase. Note that the claim refers only to columns k and ℓ in the d-by-d matrix considered in Item 1, and that $\Delta_{2,(k,\ell)}$ is the set of unsorted rows.

Let Q denote a d-by-2 zero-one matrix in which each column is sorted. Let o_1 (resp., o_2) denote the number of ones in the first (resp., second) column of Q. Then, the number of unsorted rows in Q is $r(Q) \stackrel{\text{def}}{=} o_1 - o_2$ if $o_1 > o_2$ and $r(Q) \stackrel{\text{def}}{=} 0$ otherwise. Let Q' be any matrix with o_1 (resp., o_2) 1's in its first (resp., second) column. That is, Q' is such that if we sort its columns we obtain Q. Then we claim that the number of unsorted rows in Q' is at least $r(Q)$. The claim is obvious in case $r(Q)=0$. In case $r(Q)>0$ we consider the location of the o_1 1's in the first column of Q'. At most o_2 of the corresponding entries in the second column are also 1 (as the total of 1's in the second row is o_2), and so the remaining rows (which are at least $o_1 - o_2$ in number) are unsorted. ∎

With Lemma 8 at our disposal, we are ready to state and prove that the analysis of Algorithm 2 (for any n) reduces to its analysis in the special case $n=1$.

Lemma 9. *Let A denote a single iteration of Algorithm 2, and $f:\Sigma^n \to \{0,1\}$. Then there exists functions $f_{i,\alpha,\beta}:\Sigma\to\{0,1\}$, for $i\in[n]$, $\alpha\in\{0,1\}^{i-1}$ and $\beta\in\{0,1\}^{n-i}$, so that the following holds*

1. $\epsilon_{\mathrm{M}}(f) \le 2\cdot\sum_i \mathrm{E}_{\alpha,\beta}(\epsilon_{\mathrm{M}}(f_{i,\alpha,\beta}))$, *where the expectation is taken uniformly over $\alpha\in\{0,1\}^{i-1}$ and $\beta\in\{0,1\}^{n-i}$.*
2. *The probability that A rejects f is lower bounded by the expected value of* $\mathrm{Prob}[A \text{ rejects } f_{i,\alpha,\beta}]$, *where the expectation is taken uniformly over $i\in[n]$, $\alpha\in\{0,1\}^{i-1}$ and $\beta\in\{0,1\}^{n-i}$.*

In fact, Theorem 1 follows easily from the above lemma, since in the binary case Algorithm 2 collapses to Algorithm 1 (as there is only one possible distribution p – the one assigning all weight to the single admissible pair $(1,2)$). Also, in the binary case, for any $f':\{0,1\}\to\{0,1\}$, algorithm A rejects with probability exactly $2\epsilon_{\mathrm{M}}(f')$. Thus, the lemma implies that in the binary case, for any $f:\{0,1\}^n\to\{0,1\}$, algorithm A rejects with probability at least

$$\mathrm{E}_{i,\alpha,\beta}(\mathrm{Prob}[A \text{ rejects } f_{i,\alpha,\beta}]) = \mathrm{E}_{i,\alpha,\beta}(2\epsilon_{\mathrm{M}}(f_{i,\alpha,\beta}))$$

$$= \frac{2}{n} \cdot \sum_i \mathrm{E}_{\alpha,\beta}(\epsilon_{\mathrm{M}}(f_{i,\alpha,\beta}))$$
$$\geq \frac{1}{n} \cdot \epsilon_{\mathrm{M}}(f)$$

The application of the above lemma in the non-binary case is less straightforward (as there the probability that A rejects $f' : \Sigma \to \{0,1\}$ is not necessarily $2\epsilon_{\mathrm{M}}(f')$). Furthermore, algorithm A may be one of infinitely many possibilities, depending on the infinitely many possible distributions p. But let us first prove the lemma.

Proof. For $i = 1, \ldots, n+1$, we define $f_i \stackrel{\mathrm{def}}{=} \mathrm{S}_{i-1} \cdots \mathrm{S}_1(f)$. Thus, $f_1 \equiv f$, and by Item 1 of Lemma 8, we have that f_{n+1} is monotone. It follows that

$$\epsilon_{\mathrm{M}}(f) \leq \mathrm{dist}(f, f_{n+1}) \leq \sum_{i=1}^{n} \mathrm{dist}(f_i, f_{i+1}) \tag{11}$$

Next, for $i = 1, \ldots, n$, $\alpha \in \{0,1\}^{i-1}$ and $\beta \in \{0,1\}^{n-i}$, define the function $f_{i,\alpha,\beta} : \Sigma \to \{0,1\}$, by $f_{i,\alpha,\beta}(x) = f(\alpha x \beta)$, for $x \in \Sigma$. Throughout the proof, $\sum_{\alpha,\beta}$ refers to summing over all (α,β)'s in $\Sigma^{i-1} \times \Sigma^{n-i}$, and $\mathrm{E}_{\alpha,\beta}$ refers to expectation over uniformly distributed $(\alpha,\beta) \in \Sigma^{i-1} \times \Sigma^{n-i}$. We claim that

$$\mathrm{dist}(f_i, f_{i+1}) \leq 2 \cdot \mathrm{E}_{\alpha,\beta}(\epsilon_{\mathrm{M}}(f_{i,\alpha,\beta})) \tag{12}$$

This inequality is proven (below) by observing that f_{i+1} is obtained from f_i by sorting, separately, the elements in each $f_{i,\alpha,\beta}$. (The factor of 2 is due to the relationship between the distance of a vector to its sorted form and its distance to monotone.) Thus,

$$\begin{aligned} d^n \cdot \mathrm{dist}(f_i, f_{i+1}) &= \sum_{\alpha,\beta} |\{x \in \Sigma : f_i(\alpha\, x\, \beta) \neq f_{i+1}(\alpha\, x\, \beta)\}| \\ &= \sum_{\alpha,\beta} |\{x \in \Sigma : f_{i,\alpha,\beta}(x) \neq f_{i+1,\alpha,\beta}(x)\}| \\ &= \sum_{\alpha,\beta} |\{x \in \Sigma : f_{i,\alpha,\beta}(x) \neq \mathrm{S}_i(f_{i,\alpha,\beta})(x)\}| \\ &\leq \sum_{\alpha,\beta} 2d \cdot \epsilon_{\mathrm{M}}(f_{i,\alpha,\beta}) \end{aligned}$$

where the inequality is justified as follows. Consider a vector $v \in \{0,1\}^d$ (representing a generic $f_{i,\alpha,\beta}$), and let $\mathrm{S}(v)$ denote its sorted version. Then $\mathrm{S}(v) = 0^z 1^{d-z}$, where z denotes the number of zeros in v. Thus, for some $e \geq 0$, the vector v has e 1-entries within its prefix of length z and e 0-entries in its suffix of length $(d-z)$. So the number of locations on which v and

$\mathrm{S}(v)$ disagree is exactly $2e$. On the other hand, consider an arbitrary perfect matching of the e 1-entries in the prefix and the e 0-entries in the suffix. To make v monotone one must alter at least one entry in each matched pair; thus, $\epsilon_{\mathrm{M}}(v) \geq e/d$. Equation (12) follows.

Combining Equations (11) and (12), the first item of the lemma follows. In order to prove the second item, we use the definition of algorithm A and let $\chi(E)=1$ if E holds and $\chi(E)=0$ otherwise.

$$\begin{aligned}
\mathrm{Prob}[A \text{ rejects } f] &= \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{\alpha\,,\,\beta} \mathrm{Prob}_{(k,\ell)\sim p}[f(\alpha\, k\, \beta) > f(\alpha\, \ell\, \beta)] \\
&= \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{\alpha\,,\,\beta} \sum_{(k,\ell)} p(k,\ell) \cdot \chi[f(\alpha\, k\, \beta) > f(\alpha\, \ell\, \beta)] \\
&= \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{(k,\ell)} p(k,\ell) \cdot \sum_{\alpha\,,\,\beta} \chi[f(\alpha\, k\, \beta) > f(\alpha\, \ell\, \beta)] \\
&= \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{(k,\ell)} p(k,\ell) \cdot |\Delta_{i,(k,\ell)}(f)|
\end{aligned}$$

Using Item 2 of Lemma 8, we have

$$\begin{aligned}
|\Delta_{i,(k,\ell)}(f)| &\geq |\Delta_{i,(k,\ell)}(\mathrm{S}_{i-1}(f))| \\
&\cdots \\
&\geq |\Delta_{i,(k,\ell)}(\mathrm{S}_{i-1} \cdots \mathrm{S}_1(f))|
\end{aligned}$$

Combining the above with the definition of f_i, we have

$$\begin{aligned}
\mathrm{Prob}[A \text{ rejects } f] &\geq \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{(k,\ell)} p(k,\ell) \cdot |\Delta_{i,(k,\ell)}(f_i)| \\
&= \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{(k,\ell)} p(k,\ell) \cdot \sum_{\alpha\,,\,\beta} \chi[f_i(\alpha\, k\, \beta) > f_i(\alpha\, \ell\, \beta)] \\
&= \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{\alpha\,,\,\beta} \sum_{(k,\ell)} p(k,\ell) \cdot \chi[f_{i,\alpha,\beta}(k) > f_{i,\alpha,\beta}(\ell)] \\
&= \frac{1}{n \cdot d^{n-1}} \sum_{i=1}^{n} \sum_{\alpha\,,\,\beta} \mathrm{Prob}[A \text{ rejects } f_{i,\alpha,\beta}] \\
&= \mathrm{E}_{i\,,\,\alpha\,,\,\beta}\,(\mathrm{Prob}[A \text{ rejects } f_{i,\alpha,\beta}])
\end{aligned}$$

and the lemma follows. ∎

5.1.2. Algorithms for the case $n = 1$. By the above reduction (i.e., Lemma 9), we may focus on designing algorithms for the case $n=1$. The design of such algorithms amounts to the design of a probability distribution $p:\Sigma^2 \to [0,1]$ (with support only on pairs (k,ℓ) with $k<\ell$), and the specification of the number of times that the basic iteration of Algorithm 2 is performed. We present three such algorithms, and analyze the performance of a single iteration in them.

Algorithm 2.1. This algorithm uses the uniform distribution over pairs $(k,k+1)$, and $t(n,\epsilon,d) = O(nd/\epsilon)$. That is, it uses the distribution $p_1 : \Sigma \times \Sigma \to [0,1]$ defined by $p_1(k,k+1)=1/(d-1)$ for $k=1,\ldots,d-1$.

Proposition 10. *Let A_1 denote a single iteration of Algorithm 2.1, and $f':\Sigma \to \{0,1\}$. Then, the probability that A_1 rejects f' is at least $\frac{2}{d-1}\cdot\epsilon_{\mathrm{M}}(f')$.*

The lower bound can be shown to be tight (by considering the function f' defined by $f'(x)=1$ if $x<d/2$ and $f(x)=0$ otherwise).

Proof. If $\epsilon_{\mathrm{M}}(f')>0$ then there exists a $k\in\{1,\ldots,d-1\}$ so that $f'(k)=1$ and $f(k+1)=0$. In such a case A_1 rejects with probability at least $1/(d-1)$. On the other hand, $\epsilon_{\mathrm{M}}(f')\le 1/2$, for every $f':\Sigma\to\{0,1\}$ (by considering the distance to either the all-zero or the all-one function). ∎

Algorithm 2.2. This algorithm uses a distribution $p_2:\Sigma\times\Sigma\to[0,1]$ that is uniform on a set P to be defined below, and $t(n,\epsilon,d)=O((n\log d)/\epsilon)$. The set P consists of pairs (k,ℓ), where $0<\ell-k\le 2^t$ and 2^t is the largest power of 2 that divides either k or ℓ. That is, let $\mathrm{power}_2(i)\in\{0,1\ldots,\log_2 i\}$ denote the largest power of 2 that divides i. Then,

$$P \stackrel{\text{def}}{=} \{(k,\ell)\in\Sigma\times\Sigma : 0<\ell-k\le 2^{\max(\mathrm{power}_2(k),\mathrm{power}_2(\ell))}\} \tag{13}$$

We note that selecting a pair uniformly in P can be approximated by selecting a pair $(a,b)\in\Sigma\times\Sigma$ (where either $a<b$ or $b<a$, and $\mathrm{power}_2(a)>\mathrm{power}_2(b)$) according to the following process. First, uniformly select $i\in\{1,\ldots,\lfloor\log d\rfloor\}$. Next, uniformly select $a\in\Sigma$ such that $\mathrm{power}_2(a)=i$. Finally select b uniformly in $\{a-2^{\mathrm{power}_2(a)}+1, a+2^{\mathrm{power}_2(a)}-1\}\cap(\Sigma\setminus\{a\})$. The probability of selecting any pair differs by at most a factor of 2 from that induced by the uniform distribution on P. It is not hard to verify that this suffices for our purposes.

We also mention that an algorithm of similar performance was presented and analyzed in [21, Sec. 2.1]. Loosely speaking, their algorithm selects a pair

(k,ℓ) by first picking k uniformly in $\{1,\ldots,d-1\}$, next selects t uniformly in $\{0,1,\ldots,\log_2(d-k)\}$, and finally selects ℓ uniformly in $\{k+1,\ldots,k+2^t\}\cap\Sigma$.[7]

Proposition 11. *Let A_2 denote a single iteration of Algorithm 2.2, and $f':\Sigma\to\{0,1\}$. Then, the probability that A_2 rejects f' is at least $\Omega(\frac{1}{\log d})\cdot\epsilon_{\mathrm{M}}(f')$.*

Proof. We first show that $|P|=O(d\log d)$. This can be shown by charging each pair $(k,\ell)\in P$ to the element divisible by the larger power of 2 (i.e., to k if $\mathrm{power}_2(k)>\mathrm{power}_2(\ell)$ and to ℓ otherwise), and noting that the charge incurred on each i is at most $2\cdot 2^{\mathrm{power}_2(i)}$. It follows that the total charge is at most $\sum_{i=1}^{d}2^{\mathrm{power}_2(i)+1}=\sum_{j=0}^{\log_2 d}\frac{d}{2^j}\cdot 2^{j+1}=O(d\log d)$.

We say that a pair $(k,\ell)\in P$ (where $k<\ell$) is a *violating pair* (with respect to f'), if $f'(k)>f'(\ell)$. By definition, the probability that A_2 rejects f' is the ratio between the number of violating pairs in P (with respect to f'), and the size of P. Thus, it remains to show that the former is $\Omega(\epsilon_{\mathrm{M}}(f)\cdot d)$.

In the following argument it will be convenient to view the indices $1,\ldots,d$ as vertices of a graph and the pairs in P as edges. Specifically, each pair (k,ℓ), where $k<\ell$ corresponds to a *directed* edge from k to ℓ. We refer to this graph as G_P.

Claim 11.1. For every two vertices k and ℓ in G_P, if $k<\ell$ then there is a directed path of length at most 2 from k to ℓ in G_P.

Proof of Claim. Let $r=\lceil\log d\rceil$, and consider the binary strings of length r representing k and ℓ. Let $k=(x_{r-1},\ldots,x_0)$ and $\ell=(y_{r-1},\ldots,y_0)$. Let t be the highest index such that $x_t=0$ and $y_t=1$. Note that $x_i=y_i$ for $t<i<r$. We claim that the vertex $m=(x_{r-1},\ldots,x_{t+1},1,0,\ldots 0)$ is on a path of length 2 from k to ℓ. This follows from the definition of P, since m is divided by 2^t, while both $m-k=2^t-\sum_{i=0}^{t-1}x_i 2^i\le 2^t$ and $\ell-m=\sum_{i=0}^{t-1}y_i 2^i<2^t$. ∎

We now use the claim to provide a lower bound on the number of violating pairs. Let $z=|\{k:f'(k)=0\}|$. In what follows we think of f' as being a string of length d: $f(1)\cdots f(d)$. Then, the number of 1's in the prefix of length z of f' must equal the number of 0's in its suffix of length $(d-z)$. Let us denote this number by a, and by definition of $\epsilon_{\mathrm{M}}(f')$ we have $\epsilon_{\mathrm{M}}(f')\le 2a/d$. Consider a matching of the a 1's in the prefix of length z of f' to the a 0's in its suffix of length $(d-z)$. By the above claim, there is path of length at most 2 in G_P between every matched pair. Clearly, these paths (being of length 2) are edge-disjoint. Since each path starts at a vertex of value 1 and ends at a vertex of value 0, it must contain an edge that corresponds

[7] Observe that the two distributions are actually very different. In particular, while our distribution puts no weight on pairs (k,ℓ) such that both k and ℓ are odd, the distribution of [21] gives such pairs a total weight of almost $1/4$.

to a violating pair. Thus, we obtain $a \geq \epsilon_{\mathrm{M}}(f')d/2$ violating pairs, and the proposition follows. ∎

Algorithm 2.3. This algorithm uses the uniform distribution over all admissible pairs, and $t(n,\epsilon,d) = \min\{O(nd/\epsilon), O(n/\epsilon)^2\}$. That is, it uses the distribution $p_3 : \Sigma\times\Sigma \to [0,1]$ defined by $p_3(k,\ell) = 2/((d-1)d)$ for $1 \leq k < \ell \leq d$.

Proposition 12. *Let A_3 denote a single iteration of Algorithm 2.3, and $f' : \Sigma \to \{0,1\}$. Then, the probability that A_3 rejects f' is at least $\epsilon_{\mathrm{M}}(f')^2/2$.*

The lower bound is tight up to a constant factor: For any integer $e < d/2$, consider the function $f'(x) = 0$ if $x \in \{e+1, \ldots, 2e\}$ and $f'(x) = 1$ otherwise (then $\epsilon_{\mathrm{M}}(f') = e/d$ and A_3 rejects f' if and only if it selects a pair in $\{1,\ldots,e\}\times\{e+1,\ldots,2e\}$, which happens with probability $e^2/((d-1)d/2) \approx 2\epsilon_{\mathrm{M}}(f')^2$). On the other hand, note that if $\epsilon_{\mathrm{M}}(f') > 0$ then $\epsilon_{\mathrm{M}}(f') \geq 1/d$ and so the rejection probability is at least $\epsilon_{\mathrm{M}}(f')/2d$. This bound is also tight up to a constant factor (e.g., consider $f'(x) = 0$ if $x = 2$ and $f(x) = 1$ otherwise, then $\epsilon_{\mathrm{M}}(f') = 1/d$ and A_3 rejects f' if and only if it selects the pair $(1,2)$).

Proof. As in the proof of Proposition 11, let z be the number of zeroes in f' and let $2e$ be the number of mismatches between f' and its sorted form. Then $\epsilon_{\mathrm{M}}(f') \leq 2e/d$. On the other hand, considering the e 1-entries in the prefix of length z of f' and the e 0-entries in its suffix of length $(d-z)$, we lower bound the rejection probability by $e^2/((d-1)d/2) > 2(e/d)^2$. Combining the two, we conclude that A_3 rejects f' with probability at least $2\cdot(\epsilon_{\mathrm{M}}(f')/2)^2$. ∎

On the semi-optimality of Algorithm 2.2. We call an algorithm, within the framework of Algorithm 2, *smooth* if the number of repetitions (i.e., $t(n,d,\epsilon)$) is linear in ϵ^{-1}. Note that Algorithm 2.2 is smooth, whereas Algorithm 2.3 is not. We claim that Algorithm 2.2 is optimal in its dependence on d, among all smooth algorithms. The following argument is due to Michael Krivelevich.

Proposition 13. *Let $p : \Sigma \times \Sigma \mapsto [0,1]$ be a distribution with support only on pairs (k,ℓ) such that $k < \ell$, and ρ be such that for every non-monotone $f' : \Sigma \mapsto \{0,1\}$ it holds that*

$$\mathrm{Prob}_{(k,\ell)\sim p}[f'(k) > f'(\ell)] \geq \rho \cdot \epsilon_M(f').$$

Then $\rho \leq \frac{2}{\log_2 d}$.

Proof. The key observation is that for any consecutive $2a$ indices, p has to assign a probability mass of at least $\rho \cdot a/d$ to pairs (k,ℓ) where k is among the lowest a indices and ℓ among the higher a such indices. This observation is proven as follows. Let L, H be the low and high parts of the interval in question; that is, $L=\{s+1,\ldots,s+a\}$ and $H=\{s+a+1,\ldots,s+2a\}$, for some $s\in\{0,\ldots,d-2a\}$. Consider the function f' defined by $f'(i)=1$ if $i\in L\cup\{s+2a+1,\ldots,d\}$ and $f'(i)=0$ otherwise. Then $\epsilon_{\mathrm{M}}(f')=a/d$. On the other hand, the only pairs (k,ℓ) with $f'(k)>f'(\ell)$, are those satisfying $k\in L$ and $\ell\in H$. Thus, by definition of ρ, it must hold that $\rho\le\Pr_{(k,\ell)\sim p}[k\in L \;\&\; \ell\in H]/(a/d)$, and the observation follows.

The rest of the argument is quite straightforward: Consider $\log_2 d$ partitions of the interval $[1,d]$, so that the i^{th} partition is into consecutive segments of length 2^i. For each segment in the i^{th} partition, probability p assigns a probability mass of at least $2^{i-1}\rho/d$ to pairs where one element is in the low part of the segment and the other element is in the high part. Since these segments are disjoint and their number is $d/2^i$, it follows that p assigns a probability mass of at least $\rho/2$ to pairs among halves of segments in the i^{th} partition. These pairs are disjoint from pairs considered in the other partitions and so we conclude that $(\log_2 d)\cdot\frac{\rho}{2}\le 1$. The proposition follows. ∎

5.1.3. Conclusions for general n. Combining Lemma 9 with Propositions 11 and 12, we obtain.

Theorem 14. *Algorithm 2.2 and Algorithm 2.3 constitute testers of monotonicity for mappings $\Sigma^n\mapsto\{0,1\}$.*

- *The query complexity of Algorithm 2.2 is $O((n\log d)/\epsilon)$.*
- *The query complexity of Algorithm 2.3 is $O(n/\epsilon)^2$.*

Both algorithms run in time $O(q(n,d,\epsilon)\cdot n\log d)$, where $q(n,d,\epsilon)$ is their query complexity.

Proof. Both algorithms always accept monotone functions, and have complexities as stated. For $a=2,3$, let $\delta_a(f)$ denote the rejection probability of a single iteration of Algorithm 2.a when given access to a function $f:\Sigma^n\to\{0,1\}$. Combining Lemma 9 and Proposition 11, we have

$$\begin{aligned}\delta_2(f) &\ge \mathrm{E}_{i,\alpha,\beta}(\delta_2(f_{i,\alpha,\beta})) && \text{[By Part 2 of the lemma]}\\ &\ge \mathrm{E}_{i,\alpha,\beta}(\epsilon_{\mathrm{M}}(f_{i,\alpha,\beta})/O(\log d)) && \text{[By the proposition]}\\ &\ge \frac{\epsilon_{\mathrm{M}}(f)/O(\log d)}{2n} && \text{[By Part 1 of the lemma]}\end{aligned}$$

which establishes the claim for Algorithm 2.2. Combining Lemma 9 and Proposition 12, we have

$$\begin{aligned}\delta_3(f) &\ge \mathrm{E}_{i,\alpha,\beta}(\delta_3(f_{i,\alpha,\beta})) && \text{[By Part 2 of the lemma]}\\ &\ge \mathrm{E}_{i,\alpha,\beta}(\epsilon_{\mathrm{M}}(f_{i,\alpha,\beta})^2/2) && \text{[By the proposition]}\\ \text{where}\quad & \mathrm{E}_{i,\alpha,\beta}(\epsilon_{\mathrm{M}}(f_{i,\alpha,\beta}))\ge\epsilon_{\mathrm{M}}(f)/2n && \text{[By Part 1 of the lemma]}\end{aligned}$$

So $\delta_3(f)$ is lower bounded by the minimum of $\frac{1}{N}\cdot\sum_{j=1}^{N}x_j^2$ subject to $\frac{1}{N}\cdot\sum_{j=1}^{N}x_j\geq\epsilon_{\mathrm{M}}(f)/2n$. The minimum is obtained when all x_j's are equal, and this establishes the claim for Algorithm 2.3. ∎

5.2. General ranges

Suppose we have an algorithm for testing monotonicity of functions $f:\Sigma^n\to\{0,1\}$ (where Σ is not necessarily $\{0,1\}$). Further assume (as is the case for all algorithms presented here), that the algorithm works by selecting *pairs* of strings according to a particular distribution on pairs, and verifying that monotonicity is not violated on these pairs. We show how to extend such algorithms to functions $f:\Sigma^n\to\Xi$ while losing a factor of $|\Xi|$. We note that an alternative proof for the case $\Sigma=\{0,1\}$, which is based on a previous analysis of our testing algorithm [25], was given by Batu [10].

Without loss of generality, let $\Xi=\{0,\ldots,b\}$. The definition of ϵ_{M} extends in the natural way to functions $f:\Sigma^n\to\{0,1,\ldots,b\}$. Given a function $f:\Sigma^n\to\{0,1,\ldots,b\}$, we define Boolean functions $f_i:\Sigma^n\to\{0,1\}$, by letting $f_i(x)\stackrel{\text{def}}{=}1$ if $f(x)\geq i$ and $f_i(x)\stackrel{\text{def}}{=}0$ otherwise, for $i=1,\ldots,b$. For any algorithm A that tests monotonicity of Boolean functions as restricted above, and for any Boolean function f, let $\delta_{\mathrm{M}}^{A}(f)$ be the probability that the algorithm observes a violation when selecting a single pair according to the distribution on pairs it defines. For $f:\Sigma^n\to\{0,1,\ldots,b\}$, let $\delta_{\mathrm{M}}^{A}(f)$ be defined analogously.

Lemma 15. *Let $f:\Sigma^n\to\{0,\ldots,b\}$, and let f_i's be as defined above.*

1. $\epsilon_{\mathrm{M}}(f)\leq\sum_{i=1}^{b}\epsilon_{\mathrm{M}}(f_i)$.
2. $\delta_{\mathrm{M}}^{A}(f)\geq\delta_{\mathrm{M}}^{A}(f_i)$, *for every i.*

Combining the two items and using the relationship between δ_{M}^{A} and ϵ_{M} in the binary case (i.e., say, $\delta_{\mathrm{M}}^{A}(f_i)\geq\epsilon_{\mathrm{M}}(f_i)/F$, where F depends on $|\Sigma|$ and n), we get

$$\delta_{\mathrm{M}}^{A}(f)\geq\max_i\{\delta_{\mathrm{M}}^{A}(f_i)\}\geq\frac{1}{b}\sum_{i=1}^{b}\delta_{\mathrm{M}}^{A}(f_i)\geq\frac{1}{b}\sum_{i=1}^{b}\frac{\epsilon_{\mathrm{M}}(f_i)}{F}\geq\frac{1}{b}\cdot\frac{\epsilon_{\mathrm{M}}(f)}{F}$$

Hence, we may apply algorithm A (designed to test monotonicity of Boolean functions over general domain alphabets), to test monotonicity of functions to arbitrary range of size $b+1$; we only need to increase the number of pairs that A selects by a multiplicative factor of b.

Proof. To prove Item 2, fix any i and consider the set of violating pairs with respect to f_i. Clearly each such pair is also a violating pair with respect to

f (i.e., if $x \prec y$ and $f_i(x) > f_i(y)$ then $f_i(x) = 1$ whereas $f_i(y) = 0$, and so $f(x) \geq i > f(y)$). Thus, any pair (x,y) that contributes to $\delta_{\mathrm{M}}^{A}(f_i)$ also contributes to $\delta_{\mathrm{M}}^{A}(f)$.

To prove Item 1, consider the Boolean monotone functions closest to the f_i's. That is, for each i, let g_i be a Boolean monotone function closest to f_i. Also, let g_0 be the constant all-one function. Now, define $g: \Sigma^n \to \{0,1,\ldots,b\}$ so that $g(x) \stackrel{\text{def}}{=} i$ if i is the largest integer in $\{0,1,\ldots,b\}$ so that $g_i(x) = 1$ (such i always exists as $g_0(x)=1$).

First note that the distance of g from f is at most the sum of the distances of the g_i's from the corresponding f_i's. This is the case since if $g(x) \neq f(x)$ then there must exists an $i \in \{1,\ldots,b\}$ so that $g_i(x) \neq f_i(x)$ (since if $g_i(x) = f_i(x)$ for all i's then $g(x)=f(x)$ follows).

Finally, we show that g is monotone (and so $\epsilon_{\mathrm{M}}(f) \leq \sum_{i=1}^{b} \epsilon_{\mathrm{M}}(f_i)$ follows). Suppose towards the contradiction that $g(x) > g(y)$ for some $x \prec y$. Let $i \stackrel{\text{def}}{=} g(x)$ and $j \stackrel{\text{def}}{=} g(y) < i$. Then by definition of g, we have $g_i(x) = 1$ and $g_i(y)=0$, which contradicts the monotonicity of g_i. ∎

6. Testing whether a function is unate

By our definition of monotonicity, a function f is monotone if, for any string, increasing any of its coordinates does not decrease the value of the function. A more general notion is that of *unate* functions. Here we focus on Boolean functions over $\{0,1\}^n$. Consider the two permutations over $\{0,1\}$: $(0,1)$ and $(1,0)$. Each of these permutations π induces a total order, denoted $<_\pi$, over $\{0,1\}$. The identity permutation $\mathrm{id}=(0,1)$ induces the standard order $0 <_{\mathrm{id}} 1$, and the permutation $\overline{\mathrm{id}}=(1,0)$ induces the order $1 <_{\overline{\mathrm{id}}} 0$.

Definition 7. A function $f : \{0,1\}^n \to \{0,1\}$ is *unate* if there exists a sequence $\overline{\pi} = \pi_1 \ldots \pi_n$ where each π_i is one of the two permutations over $\{0,1\}$, for which the following holds: For any two strings $x = x_1 \cdots x_n$, and $y = y_1 \cdots y_n$, if for every i we have $x_i \leq_{\pi_i} y_i$, then $f(x) \leq f(y)$. We say in such a case the f is *monotone with respect to* $\overline{\pi}$.

In particular, if a function is monotone with respect to the sequence $\mathrm{id},\ldots,\mathrm{id}$, then we simply say that it is a monotone function, and if a function is monotone with respect to *some* $\overline{\pi}$, then it is unate. An alternative definition is that a function is unate if there exists a string $a = a_1,\ldots,a_n \in \{0,1\}^n$ such that the function $f'(x) \stackrel{\text{def}}{=} f(x \oplus a)$ is monotone.

Similarly to the algorithms presented for testing monotonicity, which search for evidence to non-monotonicity, the testing algorithm for unateness tries to find evidence to non-unateness. However, here it does not suffice to find a pair of strings x, y that differ on the i^{th} bit such that $x \prec y$ while $f(x) > f(y)$. Instead we check whether for some index i and for *each of the two permutations* π, there is a pair of strings, (x, y) that differ only the i^{th} bit, such that $x_i <_\pi y_i$, while $f(x) > f(y)$.

Algorithm 3 (Testing Unateness). On input n, ϵ and oracle access to $f:\{0,1\}^n \to \{0,1\}$, do the following:

1. Uniformly select $m = O(n^{1.5}/\epsilon)$ strings in $\{0,1\}^n$, denoted $x^1, \ldots, x^m$, and m indices in $\{1, \ldots, n\}$, denoted $i^1, \ldots, i^m$.
2. For each selected x^j, obtain the values of $f(x^j)$ and $f(y^j)$, where y^j results from x^j by flipping the i^j-th bit.
3. If unateness is found to be violated then *reject*.
   A violation occurs, if among the string-pairs $\{x^j, y^j\}$, there exist two pairs and an index i, such that in both pairs the strings differ on the i^{th} bit, but in one pair the value of the function increases when the bit is flipped from 0 to 1, and in the other pair the value of the function increases when the bit is flipped from 1 to 0.

If no contradiction to unateness is found then *accept*.

Theorem 16. *Algorithm 3 is a testing algorithm for unateness. Furthermore, if the function is unate, then Algorithm 3 always accepts.*

The furthermore clause is obvious, and so we focus on analyzing the behavior of the algorithm on functions that are ϵ-far from unate.

6.1. Proof of Theorem 16

Our aim is to reduce the analysis of Algorithm 3 to Theorem 2. We shall use the following notation.

Notation 8. For $\overline{\pi} = \pi_1 \cdots \pi_n$ (where each π_i is a permutation over $\{0,1\}$), let $\prec_{\overline{\pi}}$ denote the partial order on strings with respect to π. Namely, $x \prec_{\overline{\pi}} y$ if and only if for every index i, $x_i \leq_{\pi_i} y_i$. Let $\epsilon_{\mathrm{M},\overline{\pi}}(f)$ denote the minimum distance between f and any function g that is monotone with respect to π, and let $\delta_{\mathrm{M},\overline{\pi}}(f)$ denote the fraction of pairs x, y that differ on a single bit such that $x \prec_{\overline{\pi}} y$ but $f(x) > f(y)$.

For any f and π, consider the function $f_{\overline{\pi}}$ defined by $f_{\overline{\pi}}(x) = f(\pi_1(x_1) \cdots \pi_n(x_n))$. Then, $\epsilon_{\mathrm{M},\overline{\pi}}(f) = \epsilon_{\mathrm{M}}(f_\pi)$ and $\delta_{\mathrm{M},\overline{\pi}}(f) = \delta_{\mathrm{M}}(f_\pi)$. Hence, as a corollary to Theorem 2, we have

Corollary 17. *For any $f:\{0,1\}^n \to \{0,1\}$, and for any sequence of permutations $\overline{\pi}$,*

$$\delta_{\mathrm{M},\overline{\pi}}(f) \geq \frac{\epsilon_{\mathrm{M},\overline{\pi}}(f)}{n}.$$

Our next step is to link $\delta_{\mathrm{M},\overline{\pi}}(f)$ to quantities that govern the behavior of Algorithm 3. For each $i \in \{1,\dots,n\}$, and permutation π over $\{0,1\}$, let $\gamma_{i,\pi}(f)$ denote the fraction, among all pairs of strings that differ on a single bit, of the pairs x,y such that x and y differ only on the i^{th} bit, $x_i <_\pi y_i$, and $f(x) > f(y)$. In other words, $\gamma_{i,\pi}(f)$ is the fraction of pairs that can serve as evidence to f not being monotone with respect to any $\overline{\pi} = \pi_1,\dots,\pi_n$ such that $\pi_i = \pi$. Note that in case f is monotone with respect to some $\overline{\pi}$, then for every i, $\gamma_{i,\pi_i}(f) = 0$. More generally, $\delta_{\mathrm{M},\overline{\pi}}(f) = \sum_{i=1}^{n} \gamma_{i,\pi_i}(f)$ holds for every $\overline{\pi}$ (since each edge contributing to $\delta_{\mathrm{M},\overline{\pi}}(f)$ contributes to exactly one $\gamma_{i,\pi_i}(f)$).

The distance of f from the set of unate functions, denoted $\epsilon_{\mathrm{U}}(f)$, is the minimum distance of f to any unate function; that is, $\epsilon_{\mathrm{U}}(f) = \min_{\pi}(\epsilon_{\mathrm{M},\overline{\pi}}(f))$. We next link the $\gamma_{i,\pi}(f)$'s to $\epsilon_{\mathrm{U}}(f)$.

Lemma 18. $\sum_{i=1}^{n} \min_{\pi}\{\gamma_{i,\pi}(f)\} \geq \frac{\epsilon_{\mathrm{U}}(f)}{n}$.

Proof. Let $\overline{\pi} = \pi_1 \dots \pi_n$ be defined as follows: $\pi_i = \operatorname{argmin}_{\pi}\{\gamma_{i,\pi}(f)\}$. The key observation is

$$\delta_{\mathrm{M},\overline{\pi}}(f) = \sum_{i=1}^{n} \gamma_{i,\pi_i}(f) = \sum_{i=1}^{n} \min_{\pi}\{\gamma_{i,\pi}(f)\}$$

where the first equality holds for any π, and the second follows from the definition of this specific π. Using the above equality and invoking Corollary 17, we have

$$\sum_{i=1}^{n} \min_{\pi}\{\gamma_{i,\pi}(f)\} = \delta_{\mathrm{M},\overline{\pi}}(f) \geq \frac{\epsilon_{\mathrm{M},\overline{\pi}}(f)}{n} \geq \frac{\epsilon_{\mathrm{U}}(f)}{n}. \qquad \blacksquare$$

For each i, let $\Gamma_{i,\pi}(f)$ be the set of all pairs of strings x,y that differ only on the i^{th} bit, where $x_i <_\pi y_i$, and $f(x) > f(y)$. Lemma 18 gives us a lower bound on the sum $\sum_i \min_{\pi}\{|\Gamma_{i,\pi}|\}$. To prove Theorem 16, it suffices to show that if we uniformly select $\Omega(n^{1.5}/\epsilon_{\mathrm{U}}(f))$ pairs of strings that differ on a single bit, then with probability at least $2/3$, for some i we shall obtain both a pair belonging to $\Gamma_{i,\mathrm{id}}(f)$ and a pair belonging to $\Gamma_{i,\overline{\mathrm{id}}}(f)$ (where $\overline{\mathrm{id}}$ is the permutation $(1,0)$). The above claim is derived from the following technical lemma, which can be viewed as a generalization of the *Birthday Paradox*.

Lemma 19. *Let* $S_1, \ldots, S_n, T_1, \ldots, T_n$ *be* disjoint *subsets of a universe* X. *For each* i, *let* $p_i \stackrel{\text{def}}{=} \frac{|S_i|}{|X|}$, *and* $q_i \stackrel{\text{def}}{=} \frac{|T_i|}{|X|}$. *Let* $\rho \stackrel{\text{def}}{=} \sum_i \min(p_i, q_i) > 0$. *Then, for some constant* c, *if we uniformly select* $2c \cdot \sqrt{n}/\rho$ *elements in* X, *then with probability at least* $2/3$, *for some* i *we shall obtain at least one element in* S_i *and one in* T_i.

To derive the claim, let $X = U$ (the set of unordered pairs of strings that differ on a single bit), $S_i = \Gamma_{i,\mathrm{id}}(f)$, and $T_i = \Gamma_{i,\overline{\mathrm{id}}}(f)$. Then by Lemma 18, $\sum_i \min(p_i, q_i) \geq \epsilon_U(f)/n$. Now, using Lemma 19, the claim (and theorem) follow. So it remains to prove Lemma 19.

Proof. Suppose, without loss of generality, that $p_i \leq q_i$, for every i. As a mental experiment, we partition the sample of elements into two parts of equal size, $c \cdot \sqrt{n}/\rho$. Let I be a random variable denoting the (set of) indices of sets S_i hit by the first part of the sample.

Claim 1. With probability at least 5/6 over the choice of the first part of the sample,

$$\sum_{i \in I} p_i \geq \frac{\rho}{\sqrt{n}} \tag{14}$$

The lemma follows from Claim 1 since, conditioned on Equation (14) holding, the probability that the second part of the sample *does not* include any element from $\bigcup_{i \in I} T_i$ is at most

$$\left(1 - \sum_{i \in I} q_i\right)^{c \cdot \sqrt{n}/\rho} \leq \left(1 - \frac{\rho}{\sqrt{n}}\right)^{c \cdot \sqrt{n}/\rho} < \frac{1}{6}$$

where the last inequality holds for an appropriate choice of c. The remainder of the proof of the lemma is thus dedicated to proving Claim 1.

Proof of Claim 1. Assume without loss of generality that the sets S_i are ordered according to size. Let $S_1, \ldots, S_k$ be *all* sets with probability weight at least $\rho/2n$ each (i.e., $p_1 \geq \ldots \geq p_k \geq \rho/2n$). Then $\sum_{i=k+1}^n p_i < \sum_{i=k+1}^n (\rho/2n) < \rho/2$. Since by definition, $\rho = \sum_i \min(p_i, q_i)$, and we have assumed that $p_i \leq q_i$ for all i, we have that $\sum_{i=1}^k p_i = \rho - \sum_{i=k+1}^n p_i > \rho/2$. In other words, the probability that a uniformly selected element from X hits $\bar{S} \stackrel{\text{def}}{=} \bigcup_{i=1}^k S_i$ is greater than $\rho/2$. Thus, if we uniformly select $c \cdot \sqrt{n}/\rho$ elements in X then we expect to hit $\bar{S}$ more than $c\sqrt{n}/2$ times. By a (multiplicative) Chernoff bound, for an appropriate choice of the constant c, the probability that a uniformly selected sample of size $c \cdot \sqrt{n}/\rho$ contains less that $4 \cdot \sqrt{n}$ elements in $\bar{S}$ is less than 1/12. In what follows we assume that the number of elements from $\bar{\mathrm{S}}$ that are selected in the first part of the sample is in fact at least $4\cdot\sqrt{n}$ (and later account for the probability that this event does not occur).

Let $\mathrm{I}' \stackrel{\mathrm{def}}{=} \mathrm{I}\cap\{1,\ldots,k\}$. That is, I' is a random variable denoting the (set of) indices of sets S_i, $i\in\{1,\ldots,k\}$ that are hit by the first part of the sample. In particular, $I'\subseteq I$ (where I is as defined at the beginning of this proof).

Claim 2. Conditioned on the first sample containing at least $4\cdot\sqrt{n}$ elements from $\bar{\mathrm{S}}$, with probability at least $11/12$ it holds that $\sum_{i\in\mathrm{I}'} p_i \geq \frac{\rho}{\sqrt{n}}$.

Let E_1 denote the event that the first sample contains at least $4\cdot\sqrt{n}$ elements from $\bar{\mathrm{S}}$, and let E_2 denote the event that $\sum_{i\in\mathrm{I}'} p_i \geq \frac{\rho}{\sqrt{n}}$. By Claim 2 and the preceding discussion, the probability that Equation 14 holds is at least $\Pr[E_1]\cdot\Pr[E_2|\,E_1]\geq(11/12)^2>5/6$, proving Claim 1. It hence remains to prove Claim 2.

Proof of Claim 2. Recall that the sample is uniformly distributed in X. Thus, for each sample element, conditioned on it belonging to $\bar{\mathrm{S}}\subset\mathrm{X}$, the element is uniformly distributed in $\bar{\mathrm{S}}$. Hence, we may lower bound the probability of the above event (i.e., E_2), when selecting $4\sqrt{n}$ elements uniformly and independently in $\bar{\mathrm{S}}$. Consider the choice of the j^{th} element from $\bar{\mathrm{S}}$, and let I'_{j-1} denote the set of indices of sets hit by the the first $j-1$ elements selected in $\bar{\mathrm{S}}$. Going for $j=1,\ldots,4\sqrt{n}$, we consider two cases.

1. In case $\sum_{i\in\mathrm{I}'_{j-1}} p_i \geq \frac{2\cdot\sum_{i=1}^k p_i}{\sqrt{n}}$, we are done since $\sum_{i=1}^k p_i \geq \frac{\rho}{2}$.
2. Otherwise (i.e., $\sum_{i\in\mathrm{I}'_{j-1}} p_i < 2\sum_{i=1}^k p_i/\sqrt{n}$), the probability that the j^{th} element belongs to $\mathrm{I}'\setminus\mathrm{I}'_{j-1}$ (i.e., it hits a set in $\{\mathrm{S}_1,\ldots,\mathrm{S}_k\}$ that was not yet hit), is at least $1-(2/\sqrt{n})\cdot\sum_{i=1}^k p_i$. But $\sum_{i=1}^k p_i \leq 1/2$ (as $p_i\leq q_i$ for all i and $\sum_{i=1}^n p_i+\sum_{i=1}^n q_i\leq 1$), and so this probability is at least $1-1/\sqrt{n}$, which is at least $2/3$ for $n\geq 9$. Since each such element carries a p_i weight of at least $\rho/2n$, it follows that with probability at least $2/3$ the sum of p_i's has increased by at least $\rho/2n$.

Observe that if we toss $4\sqrt{n}$ (or more) coins with bias $2/3$ towards heads, then with probability at least $11/12$ (provided n is big enough) we'll get at least $2\sqrt{n}$ heads. In our case, the number of coins tossed corresponds to the number of elements that are selected in $\bar{S}$, and the heads correspond to getting a new element from $\bar{\mathrm{S}}$. Thus, if Case 2 occurs $4\sqrt{n}$ (or more) times then with probability at least $11/12$ the sum $\sum_{i\in\mathrm{I}'} p_i$ is at least $2\sqrt{n}\cdot(\rho/2n)=\rho/2$, and the claim follows. (Claim 2, Claim 1, and Lemma 19.) ∎

7. Testing based on random examples

In this section we prove Theorems 5 and 6: establishing a lower bound on the sample complexity of such testers and a matching algorithm, respectively. For convenience, we first restate the theorems.

Theorem 5. *For any $\epsilon = O(n^{-3/2})$, any tester for monotonicity that only utilizes random examples must use at least $\Omega(\sqrt{2^n/\epsilon})$ such examples.*

Theorem 6. *There exists a tester for monotonicity that only utilizes random examples and uses at most $O(\sqrt{2^n/\epsilon})$ examples, provided $\epsilon > n^2 \cdot 2^{-n}$. Furthermore, the algorithm runs in time* $\mathrm{poly}(n) \cdot \sqrt{2^n/\epsilon}$.

7.1. A lower bound on sample complexity

As in the proof of Proposition 4, we view the Boolean Lattice as a directed layered graph G_n, where the i^{th} layer is denoted L_i. Consider the vertices in L_k and L_{k-1}, where $k = \lfloor (n+1)/2 \rfloor$. We know that $|\mathrm{L}_k|, |\mathrm{L}_{k-1}| = \Omega(n^{-1/2} \cdot 2^n)$. It can be shown (*cf.* [18, Chap. 2, Cor. 4]) using Hall's Theorem, that for any such pair of adjacent layers, there exists a perfect matching between the smallest among the two layers and a subset of the larger layer. Let $\mathrm{M} = \{(v_i, u_i) : i=1, \ldots, t\} \subseteq L_{k-1} \times L_k$ denote this matching, where $t = |\mathrm{L}_{k-1}|$. Using a greedy approach, we find a large matching $\mathrm{M}' \subset \mathrm{M}$, $\mathrm{M}' = \{(v_{i_j}, u_{i_j})\}$ such that there are no edges in G_n between pairs v_{i_j} and u_{i_k} such that $i_j \neq i_k$. Since each edge $(v_i, u_i) \in \mathrm{M}'$ "rules out" at most $(k-1) + (n-(k-1)-1) < n$ other edges in M (i.e., an edge (v_j, u_j) is ruled out if either (v_j, u_i) or (v_i, u_j) is an edge in G_n), we can obtain $|\mathrm{M}'| \geq \frac{t}{n} = \Omega(n^{-3/2} \cdot 2^n)$. By possibly dropping edges from M' we can obtain a matching M'' so that $|\mathrm{M}''|$ is even and of size $2\epsilon \cdot 2^n$ (recall that $\epsilon = O(n^{-3/2})$). Using M'' we define two families of functions. A function in each of the two families is determined by a partition of M'' into two sets, A and B, of equal size.

1. A function f in the first family is defined as follows
   - For every $(v, u) \in A$, define $f(v) = 1$ and $f(u) = 0$.
   - For every $(v, u) \in B$, define $f(v) = 0$ and $f(u) = 1$.
   - For x with $w(x) \geq k$, for which f has not been defined, define $f(x) = 1$.
   - For x with $w(x) \leq k-1$, for which f has not been defined, define $f(x) = 0$.
2. A function f in the second family is defined as follows
   - For every $(v, u) \in A$, define $f(v) = 1$ and $f(u) = 1$.
   - For every $(v, u) \in B$, define $f(v) = 0$ and $f(u) = 0$.

– For x's on which f has not been defined, define $f(x)$ as in the first family.

It is easy to see that every function in the second family is monotone, whereas for every function f in the first family $\epsilon_{\mathrm{M}}(f) = |B|/2^n = \epsilon$. Theorem 5 is established by showing that an algorithm which obtains $o(\sqrt{|B|})$ random examples cannot distinguish a function uniformly selected in the first family (which needs to be rejected with probability at least $2/3$) from a function uniformly selected in the second family (which needs to be accepted with probability at least $2/3$). That is, we show that the statistical distance between two such samples is too small.

Claim 20. *The statistical difference between the distributions induced by the following two random processes is bounded above by* $\binom{m}{2} \cdot \frac{|\mathrm{M}''|}{2^{2n}}$. *The first process* (resp., *second process*) *is define as follows*

- *Uniformly select a function f in the* first (resp., second) *family.*
- *Uniformly and independently select m strings, $x^1,\ldots,x^m$, in $\{0,1\}^n$.*
- *Output $(x^1, f(x^1)),\ldots,(x^m, f(x^m))$.*

Proof. The randomness in both processes amounts to the choice of B (uniform among all $(|\mathrm{M}''|/2)$-subsets of M'') and the uniform choice of the sequence of x^i's. The processes differ only in the labelings of the x^i's that are matched by M'', yet for u (resp., v) so that $(u,v)\in\mathrm{M}''$ the label of u (resp., v) is uniformly distributed in both processes. The statistical difference is due merely to the case in which for some i,j the pair (x^i,x^j) resides in M''. The probability of this event is bounded by $\binom{m}{2}$ times the probability that a specific pair (x^i,x^j) resides in M''. The latter probability equals $\frac{|\mathrm{M}''|}{2^n}\cdot 2^{-n}$. ∎

Conclusion. By Claim 20, $m<2^n/\sqrt{3|\mathrm{M}''|}$ implies that the statistical difference between these processes is less than $\frac{m^2}{2}\cdot\frac{|\mathrm{M}''|}{2^{2n}} < 1/6$ and thus an algorithm utilizing m queries will fail to work for the parameter $\epsilon=|B|/2^n$. Theorem 5 follows. ∎

7.2. A matching algorithm

The algorithm consists of merely emulating Algorithm 1. That is, the algorithm is given $m \stackrel{\text{def}}{=} O(\sqrt{2^n/\epsilon})$ uniformly selected examples and tries to find a violating pair as in Step 3 of Algorithm 1. We assume $\epsilon > n^2\cdot 2^{-n}$, or else the algorithm sets $m=O(n\cdot 2^n)$.

Algorithm 4. Input n,ϵ and $(x^1, f(x^1)),\ldots,(x^m, f(x^m))$.

1. Place all $(x^j, f(x^j))$'s on a heap arranged according to any ordering on $\{0,1\}^n$.
2. For $j=1,\ldots,m$ and $i=1,\ldots,n$, try to retrieve from the heap the value $y \stackrel{\text{def}}{=} x^j \oplus 0^{i-1}10^{n-i}$. If successful then consider the values $x^j, y, f(x^j), f(y)$ and in case they demonstrate that f is not monotone then *reject*.

If all iterations were completed without rejecting then *accept*.

Analysis. Clearly, Algorithm 4 always accepts a monotone function, and can be implemented in time $\text{poly}(n)\cdot m$. Using a Birthday Paradox argument, we show that for the above choice of $m = O(\sqrt{2^n/\epsilon})$, Algorithm 4 indeed rejects ϵ-far from monotone functions with high probability. We merely need to show the following.

Lemma 21. *There exists a constant c so that the following holds. If $m \geq c\cdot\sqrt{2^n/\epsilon_{\mathrm{M}}(f)}$ and if the x^i's are uniformly and independently selected in $\{0,1\}^n$ then Algorithm 4 rejects the function f with probability at least 2/3.*

Proof. We consider the sets U and $\Delta(f)$, as defined in the proof of Theorem 2 (see Equation (2) and Equation (3), respectively). By Theorem 2, we have $|\Delta(f)| \geq \frac{\epsilon_{\mathrm{M}}(f)}{n}\cdot|\mathrm{U}| = \epsilon_{\mathrm{M}}(f)\cdot 2^{n-1}$. Our goal is to lower bound the probability that the m-sample contains a pair in $\Delta(f)$. Towards this end, we partition the sample into two equal parts, denoted $x^{(1)},\ldots,x^{(m/2)}$ and $y^{(1)},\ldots,y^{(m/2)}$. For $i,j\in\{1,\ldots,m/2\}$, we define a 0-1 random variable $\zeta_{i,j}$ so that $\zeta_{i,j}=1$ if $(x^{(i)},y^{(j)})\in\Delta(f)$ and $\zeta_{i,j}=0$ otherwise. Clearly, the $\zeta_{i,j}$'s are identically distributed and we are interested in the probability that at least one of them equals 1 (equiv., their sum is positive). Note that the $\zeta_{i,j}$'s are dependent random variables, but they are almost pairwise independent as shown below. We first show that the expected value of their sum is at least $c^2/8$. Below, X and Y are independent random variables uniformly distributed over $\{0,1\}^n$.

$$
\begin{aligned}
(15)\qquad \mu \stackrel{\text{def}}{=} \mathrm{E}[\zeta_{i,j}] &= \Pr_{X,Y}[(X,Y)\in\Delta(f)] \\
&= \sum_{(x,y)\in\Delta(f)} \Pr_{X,Y}[X=x \,\&\, Y=y] \\
&= |\Delta(f)|\cdot\left(2^{-n}\right)^2 \geq \epsilon_{\mathrm{M}}(f)\cdot 2^{-(n+1)}
\end{aligned}
$$

Thus, $\mathrm{E}[\sum_{i,j}\zeta_{i,j}] \geq (m/2)^2\cdot\epsilon_{\mathrm{M}}(f)2^{-(n+1)} \geq c^2/8$, which for sufficiently large value of the constant c yields a big constant. It thus come at little surprise that the probability that $\sum_{i,j}\zeta_{i,j}=0$ is very small. Details follows.

Let $\overline{\zeta}_{i,j} \stackrel{\text{def}}{=} \zeta_{i,j} - \mu$. Using Chebishev's Inequality we have

$$\begin{aligned}
\Pr[\sum_{i,j} \zeta_{i,j} = 0] &\le \Pr\left[\left|\sum_{i,j} \overline{\zeta}_{i,j}\right| \ge (m/2)^2 \cdot \mu\right] \\
&\le \frac{\mathrm{E}[(\sum_{i,j} \overline{\zeta}_{i,j})^2]}{(m/2)^4 \cdot \mu^2} \\
&\le \frac{\sum_{i,j} \mathrm{E}[\overline{\zeta}^2_{i,j}]}{(m/2)^4 \cdot \mu^2} + 2 \cdot \frac{\sum_{i,j,k \text{ s.t. } j \neq k} \mathrm{E}[\overline{\zeta}_{i,j} \overline{\zeta}_{i,k}]}{(m/2)^4 \cdot \mu^2}
\end{aligned}$$

using $\mathrm{E}[\overline{\zeta}_{i,j}\overline{\zeta}_{i',j'}] = \mathrm{E}[\overline{\zeta}_{i,j}] \cdot \mathrm{E}[\overline{\zeta}_{i',j'}] = 0$, for every 4-tuple satisfying $i \neq i'$ and $j \neq j'$. (The factor of 2 compensates for the symmetric terms $\mathrm{E}[\overline{\zeta}_{i,j}\overline{\zeta}_{k,j}]$ s.t. $i \neq k$.) Since $\mathrm{E}[\overline{\zeta}^2_{i,j}] = \mathrm{E}[\zeta^2_{i,j}] - \mu^2$, the first term above is bounded by

$$\frac{\sum_{i,j} \mathrm{E}[\zeta^2_{i,j}]}{(m/2)^4 \cdot \mu^2} = \frac{\sum_{i,j} \mathrm{E}[\zeta_{i,j}]}{(m/2)^4 \cdot \mu^2} = \frac{1}{(m/2)^2 \cdot \mu} \le \frac{8}{c^2}$$

where the first equality follows from the fact that $\zeta_{i,j}$ is a zero-one random variable. To bound the second term, we let X, Y and Z be independent random variables uniformly distributed over $\{0,1\}^n$, and obtain

$$\begin{aligned}
&\sum_{i,j,k \text{ s.t. } j \neq k} \mathrm{E}[\overline{\zeta}_{i,j}\overline{\zeta}_{i,k}] \le \sum_{i,j,k \text{ s.t. } j \neq k} \mathrm{E}[\zeta_{i,j}\zeta_{i,k}] \\
&\quad \le (m/2)^3 \cdot \Pr_{X,Y,Z}[(X,Y) \in \Delta(f) \& (X,Z) \in \Delta(f)] \\
&\quad = (m/2)^3 \cdot |\{(x,y,z) : (x,y) \in \Delta(f) \& (x,z) \in \Delta(f)\}| \cdot \left(2^{-n}\right)^3 \\
&\quad \le (m/2)^3 \cdot |\{(x,y,z) : (x,y) \in \Delta(f) \& (x,z) \in \mathrm{U}\}| \cdot \left(2^{-n}\right)^3 \\
&\quad \le (m/2)^3 \cdot (|\Delta(f)| \cdot n) \cdot 2^{-3n} \\
&\quad = (m/2)^3 \cdot \mu \cdot n \cdot 2^{-n}
\end{aligned}$$

Combining all the above, we get

$$\begin{aligned}
\Pr[\sum_{i,j} \zeta_{i,j} = 0] &\le \frac{8}{c^2} + 2 \cdot \frac{(m/2)^3 \cdot \mu \cdot n \cdot 2^{-n}}{(m/2)^4 \mu^2} \\
&\le \frac{8}{c^2} + 8 \cdot \frac{n \cdot 2^{-n}}{c\sqrt{2^n/\epsilon} \cdot \epsilon 2^{-n}}
\end{aligned}$$

Using $\epsilon \ge n^2 2^{-n}$, the second term is bounded by $8/c$, and the lemma follows (for $c \ge 25$).

Acknowledgments. We would like to thank Dan Kleitmann and Michael Krivelevich for helpful discussions. In particular, Proposition 13 is due to

Michael Krivelevich. We also thank two anonymous referees for their helpful comments and corrections.

References

[1] N. Alon: On the density of sets of vectors, *Discrete Mathematics*, **46** (1983), 199–202.

[2] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy: Efficient testing of large graphs, in: *Proceedings of FOCS99*, 1999.

[3] N. Alon and M. Krivelevich: Testing k-colorability, Manuscript, 1999.

[4] D. Angluin: Queries and concept learning, *Machine Learning*, **2**(4) (1988), 319–342.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy: Proof verification and intractability of approximation problems, *JACM*, **45**(3) (1998), 501–555.

[6] S. Arora and S. Safra: Probabilistic checkable proofs: A new characterization of NP, *JACM*, **45**(1) (1998), 70–122.

[7] S. Arora and S. Sudan: Improved low degree testing and its applications, in: *Proceedings of STOC97*, (1997), 485–495.

[8] L. Babai, L. Fortnow, L. Levin, and M. Szegedy: Checking computations in polylogarithmic time, in: *Proceedings of STOC91*, (1991), 21–31.

[9] L. Babai, L. Fortnow, and C. Lund: Non-deterministic exponential time has two-prover interactive protocols, *Computational Complexity*, **1**(1) (1991), 3–40.

[10] T. Batu: An extension to testing monotonicity, Manuscript, 1998.

[11] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan: Linearity testing in characteristic two, *IEEE Transactions on Information Theory*, (1996), 1781–1795.

[12] M. Bellare, O. Goldreich, and M. Sudan: Free bits, PCPs and non-approximability – towards tight results, *SIAM Journal on Computing*, **27**(3) (1998), 804–915.

[13] M. Bellare, S. Goldwasser, C. Lund, and A. Russell: Efficient probabilistically checkable proofs and applications to approximation, in: *Proceedings of STOC93*, (1993), 294–304.

[14] M. Bellare and M. Sudan: Improved non-approximability results, in: *Proceedings of STOC94*, (1994), 184–193.

[15] M. Bender and D. Ron: Testing acyclicity of directed graphs in sublinear time, Manuscript, 1999.

[16] A. Blum, C. Burch, and J. Langford: On learning monotone Boolean functions, in: *Proceedings of FOCS98*, 1998.

[17] M. Blum, M. Luby, and R. Rubinfeld: Self-testing/correcting with applications to numerical problems, *JACM*, **47** (1993), 549–595.

[18] B. Bollobás: *Combinatorics*, Cambridge University Press, 1986.

[19] J. Bourgain, J. Kahn, G. Kalai, Y. Katznelson, and N. Linial: The influence of variables in product spaces, *Israel Journal of Mathematics*, **77** (1992), 55–64.

[20] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky: Improved testing algorithms for monotonocity, in: *Proceedings of Random99*, (1999), 97–108.

[21] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan: Spot-checkers, in: *Proceedings of STOC98*, (1998), 259–268.

[22] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy: Approximating clique is almost NP-complete, *JACM*, **43**(2) (1996), 268–292.

[23] P. Frankl: The shifting technique in extremal set theory, *Surveys in Combinatorics*, 1987, London Mathematical Society Notes in Mathematics 123, Cambridge University Press.

[24] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson: Self-testing/correcting for polynomials and for approximate functions, in: *Proceedings of STOC91*, (1991), 32–42.

[25] O. Goldreich, S. Goldwasser, E. Lehman, and D. Ron: Testing monotonicity, in: *Proceedings of FOCS98*, (1998), 426–435.

[26] O. Goldreich, S. Goldwasser, and D. Ron: Property testing and its connection to learning and approximation, *JACM*, **45**(4) (1998), 653–750.

[27] O. Goldreich and D. Ron: Property testing in bounded degree graphs, in: *Proceedings of STOC97*, (1997), 406–415.

[28] O. Goldreich and D. Ron: A sublinear bipartite tester for bounded degree graphs, *Combinatorica*, (1999), 1–39.

[29] J. Håstad: Testing of the long code and hardness for clique, in: *Proceedings of STOC96*, 11–19, 1996.

[30] J. Håstad: Getting optimal in-approximability results, in: *Proceedings of STOC97*, (1997), 1–10.

[31] M. Kearns, M. Li, and L. Valiant: Learning boolean formulae, *JACM*, **41**(6) (1994), 1298–1328.

[32] M. Kearns and L. Valiant: Cryptographic limitations on learning boolean formulae and finite automata, *JACM*, **41**(1) (1994), 67–95.

[33] M. Kiwi: *Probabilistically Checkable Proofs and the Testing of Hadamard-like Codes*, PhD thesis, MIT, 1996.

[34] M. Parnas and D. Ron: Testing the diameter of graphs, in: *Proceedings of Random99*, (1999), 85–96.

[35] R. Raz and S. Safra: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, in: *Proceedings of STOC97*, (1997), 475–484.

[36] R. Rubinfeld and M. Sudan: Robust characterization of polynomials with applications to program testing, *SIAM Journal on Computing*, **25**(2) (1996), 252–271.

[37] L. Trevisan: Recycling queries in PCPs and in linearity tests, in: *Proceedings of STOC98*, 299–308, 1998.

Appendix

Here we give counterexamples to generalizations of Item 2 in Lemma 7. Recall that this item asserts that for every 2×2 zero-one valued matrix, if the columns of the matrix are sorted, then the number of modification required to sort the rows (i.e., twice the number of unsorted rows), cannot increase. Here we show that this claim does not generalize neither to $d\times d$ zero-one matrices for $d\geq 4$ nor to 2×2 matrices over Σ such that $|\Sigma|\geq 3$.

Example 1: a 2-by-4 zero-one matrix. Consider the matrix

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

The first row is sorted, and in order to sort the second row two modifications are necessary and sufficient. However, after sorting the columns we get:

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

and now the first row requires two modifications, and so does the second row. Hence, the total number of modification required in order to sort the rows has increased following the sorting of the columns.

Example 2: a 2-by-2 3-valued matrix. Consider the matrix:

$$\begin{pmatrix} 3 & 1 \\ 2 & 2 \end{pmatrix}$$

The first row requires two modifications, and the second row is sorted. After sorting the columns we get:

$$\begin{pmatrix} 2 & 1 \\ 3 & 2 \end{pmatrix}$$

and now both rows require two modifications.

Oded Goldreich

Department of Computer Science
and Applied Mathematics
Weizmann Institute of Science
Rehovot, Israel
oded@wisdom.weizmann.ac.il

Shafi Goldwasser

Laboratory for Computer Science
MIT
545 Technology Sq.
Cambridge, MA 02139
shafi@theory.lcs.mit.edu

Eric Lehman

Laboratory for Computer Science
MIT
545 Technology Sq.
Cambridge, MA 02139
e_lehman@theory.lcs.mit.edu

Dana Ron

Department of EE – Systems
Tel Aviv University
Ramat Aviv, ISRAEL
danar@eng.tau.ac.il

Alex Samorodnitsky

School of Mathematics
Institute for Advanced Study
Olden Lane
Princeton, NJ 08540
asamor@ias.edu