Adaptive Lower Bound for Testing Monotonicity on the Line

Aleksandrs Belovs[1]
Faculty of Computing, University of Latvia, Raina bulvaris 19, Riga, Latvia.
aleksandrs.belovs@lu.lv

Abstract

In the property testing model, the task is to distinguish objects possessing some property from the objects that are far from it. One of such properties is monotonicity, when the objects are functions from one poset to another. This is an active area of research. In this paper we study query complexity of ε-testing monotonicity of a function $f\colon [n] \to [r]$. All our lower bounds are for adaptive two-sided testers.

- We prove a nearly tight lower bound for this problem in terms of r. The bound is $\Omega\big(\frac{\log r}{\log\log r}\big)$ when $\varepsilon = 1/2$. No previous satisfactory lower bound in terms of r was known.
- We completely characterise query complexity of this problem in terms of n for smaller values of ε. The complexity is $\Theta\big(\varepsilon^{-1}\log(\varepsilon n)\big)$. Apart from giving the lower bound, this improves on the best known upper bound.

Finally, we give an alternative proof of the $\Omega(\varepsilon^{-1}d\log n - \varepsilon^{-1}\log\varepsilon^{-1})$ lower bound for testing monotonicity on the hypergrid $[n]^d$ due to Chakrabarty and Seshadhri (RANDOM'13).

2012 ACM Subject Classification Theory of computation → Lower bounds and information complexity

Keywords and phrases property testing, monotonicity on the line, monotonicity on the hypergrid

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2018.31

Funding This research is supported by the ERDF project number 1.1.1.2/I/16/113.

Acknowledgements I am grateful to Eric Blais for introducing me to this problem, and for his encouragement to work on it, as well as for pointing out Ref. [18]. I would like to thank Dmitry Gavinsky, Ansis Rosmanis, and Ronald de Wolf for helpful discussions, and anonymous referees for their suggestions. The construction of Theorem 7 is due to Ronald de Wolf. Part of this work was done while visiting Institute of Mathematics of the Czech Academy of Sciences in Prague, and Centre for Quantum Technologies in Singapore. I would like to thank Dmitry Gavinsky, Pavel Pudlák, and Miklos Santha for hospitality.

1 Introduction

The framework of property testing was formulated by Rubinfeld and Sudan [19] and Goldreich et al. [16]. A property testing problem is specified by a *property* $\mathcal{P}$, which is a class of functions mapping some finite set D into some finite set R, and *proximity parameter* ε, which is a real number between 0 and 1. An *ε-tester* is a bounded-error randomised query algorithm which, given oracle access to a function $f\colon D \to R$, distinguishes between the case when f belongs to $\mathcal{P}$ and the case when f is ε-far from $\mathcal{P}$. The latter means that any function $g \in \mathcal{P}$ differs from f on at least ε fraction of the points in the domain D. The usual complexity measure

[1] supported by the ERDF project number 1.1.1.2/I/16/113

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2018).
Editors: Eric Blais, Klaus Jansen, José D. P. Rolim, and David Steurer; Article No. 31; pp. 31:1–31:10

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is the number of queries to the function f. A tester is with *1-sided* error if it always accepts a function f in $\mathcal{P}$. A tester is *non-adaptive* if its queries do not depend on the responses received to the previous queries. The most general tester is adaptive with 2-sided error, which we will implicitly assume in this paper.

When both the domain D and the range R are partially ordered sets, a natural property to consider is that of *monotonicity*. A function $f\colon D \to R$ is called *monotone* if $x \leq y$ implies $f(x) \leq f(y)$ for all $x, y \in D$. Usually it is assumed that R is a totally ordered set. In this case, the property $\mathcal{P}$ consists of all monotone functions from D to R. The problem of testing monotonicity was explicitly formulated and studied by Goldreich et al. [15] in the case when D is the Boolean hypercube $\{0,1\}^d$ and $R = \{0,1\}$. This is an active area of research as exemplified by the papers [11, 14, 7, 6, 9, 8, 17, 1, 2, 10].

However, slightly earlier than Goldreich et al., Ergün et al. studied the same problem for functions $f\colon [n] \to [r]$. Ergün et al. called it *spot-checker for sorting*, but now this problem is generally known as *monotonicity testing on the line*, the line being the totally ordered set $[n]$. This problem is the main focus of this paper. Although this problem is arguably simpler than testing monotonicity on the hypercube, it seems more natural and important from the practical point of view. Ergün et al. mention that their algorithm can be used in software quality assurance by providing a very fast verification procedure that checks whether a presumably sorted array is indeed sorted.

A related problem is that of testing monotonicity on the *hypergrid*. A hypergrid is a set $[n]^d$ of d-tuples with elements in $[n]$. For two d-tuples $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$, we have $x \leq y$ iff $x_i \leq y_i$ for all i. We are interested in functions from $[n]^d$ to $[r]$ for some positive integers n, d and r. Clearly, this is a generalisation of both monotonicity testing on the line (when $d = 1$) and on the hypercube (when $n = 2$). These problems are closely related, so it is important to consider all of them when discussing prior work.

1.1 Prior work

We proceed with a brief discussion of previous results related to our paper. Let us start with the upper bounds. As mentioned above, the problem of testing monotonicity on the line was first considered by Ergün et al. [12], who gave an ε-tester with complexity $O\big(\varepsilon^{-1} \log n\big)$. Concerning the case of functions from the hypercube $\{0,1\}^d$ to arbitrary $[r]$, Goldreich et al. [15] proposed the *edge tester*, and Chakrabarty and Seshadhri [7] proved that it has query complexity $O(d/\varepsilon)$. In the latter paper, an ε-tester for testing monotonicity on the hypergrid $[n]^d$ with complexity $O(\varepsilon^{-1} d \log n)$ was also constructed.

Now let us turn to the lower bounds. Ergün et al. [12] proved a lower bound[2] of $\Omega(\log n)$ for testing monotonicity on the line in the so-called *comparison-based* model. In this model, each query of the tester may depend only on the order relations between the responses to the previous queries, but not on the values of the responses themselves.[3] Fischer [13] proved that any lower bound for a monotonicity testing problem in the comparison-based model implies the same lower bound in the usual value-based model. This immediately gives an $\Omega(\log n)$ lower bound for testing monotonicity on the line, matching the upper bound by Ergün et al. [12]. Chakrabarty and Seshadhri [5] used the same technique to prove a lower bound of

[2] If a lower bound does not state dependence on ε, it is assumed that the lower bound holds for some choice of $\varepsilon = \Omega(1)$.

[3] Note that this does *not* mean that the tester asks queries of the form $f(x) \stackrel{?}{\leq} f(y)$. The query is still an input x, and the tester learns about the order relations between $f(x)$ and $f(y)$ for *all* previously queried y's.

$\Omega(\varepsilon^{-1}d\log n - \varepsilon^{-1}\log\varepsilon^{-1})$ for testing monotonicity on hypergrids. Unfortunately, Fischer's construction is based on Ramsey theory, which means that, in order for this construction to work, the size of the range, r, has to be really huge.

Large values of r in the above lower bounds may lead one to study complexity of testing monotonicity with respect to the size of the range, r, rather than the size of the domain, n. Moreover, there are two parameters related to the output of the function f: the size of the range (codomain) r, and the size of the image $t = \big|f([n])\big|$: the number of different values attained by the function. Clearly, $t \le r$. This means it is more interesting to prove upper bounds in terms of t and lower bounds in terms of r.

Let us start with the upper bounds. First, it is easy to see that if $r = 2$, then $O(1/\varepsilon)$ queries suffice to ε-test monotonicity on the line. Generalising this observation, Pallavoor et al. [18] constructed an ε-tester for monotonicity on the line with complexity $O(\frac{1}{\varepsilon}\log t)$ for general t, as well as an ε-tester for monotonicity on hypergrids with complexity $O\big(\frac{d}{\varepsilon}\log\frac{d}{\varepsilon}\log t\big)$. Since $t \le n$, the lower bound of $\Omega(\log n)$ due to Fischer [13] implies the lower bound of $\Omega(\log t)$ for all $n \ge t$ and r large enough.[4]

The main prior technique capable of proving strong lower bounds for functions with small range is that of communication complexity. Blais et al. introduced this technique in [3], where it was proven that $\Omega\big(\min\{d, r^2\}\big)$ queries are required to test a function $f\colon \{0,1\}^d \to [r]$ for monotonicity. In a subsequent paper [4] a *non-adaptive* lower bound of $\Omega(d\log n)$ was proven for functions $f\colon [n]^d \to [nd]$ on the hypergrid. In the special case of $d = 1$, this gives a lower bound of $\Omega\big(\log\min\{n, r\}\big)$ for testing monotonicity of a function $f\colon [n] \to [r]$ on the line.

1.2 Our results

Our main contribution is an adaptive lower bound for testing monotonicity on the line. In order not to obstruct our main argument, we first prove the bound for $\varepsilon = \Omega(1)$, and then show how to adapt the construction for smaller values of ε.

▶ **Theorem 1.** *Every adaptive bounded-error $1/2$-tester for monotonicity of a function $f\colon [2^k] \to [k^{3k}]$ has query complexity $\Omega(k)$.*

Unlike [13, 5], we bypass Ramsey theory and construct two explicit distributions of functions that are hard to distinguish by an adaptive algorithm. In terms of the size of the domain, n, this gives the same lower bound of $\Omega(\log n)$ as in Fischer's paper [13], but with vastly reduced range size. A more direct construction can be beneficial for generalisation to other models, like quantum testers, since it is not known how to adapt Fischer's technique to the quantum settings.

In terms of the size of the range, r, this gives a lower bound of of $\Omega\big(\frac{\log r}{\log\log r}\big)$, thus nearly matching the upper bound by Pallavoor et al. [18]. Finally, we get a slightly worse estimate (in terms of the range size) than that of Blais et al. [4] but for *adaptive* testers. We prove Theorem 1 in Section 3.

In Section 4, we consider the case of general ε. First, we show that the construction of Theorem 1 can be used to prove an $\Omega\big(\varepsilon^{-1}\log(\varepsilon n)\big)$ lower bound for ε-testing monotonicity on the line. For large values of ε, this matches the upper bound of $O\big(\varepsilon^{-1}\log n\big)$ due to Ergün et al. [12], but is slightly worse when ε is close to $1/n$. However, we manage to improve the algorithm and prove an upper bound of $O\big(\varepsilon^{-1}\log(\varepsilon n)\big)$ for all $\varepsilon n \ge 2$, thus matching our lower bound. If $\varepsilon n < 2$, the complexity is obviously $\Theta(n)$, hence, this completely resolves the problem of ε-testing monotonicity on the line for all values of n and $\varepsilon \le 1/2$.

[4] The size of the domain of a function $f\colon [n] \to [r]$ can be inflated without changing its distance to monotonicity by replacing f with the function $f'\colon [nk] \to [r]$ given by $f'(x) = f(\lfloor x/k \rfloor)$.

Finally, in Section 5, we show how our construction can be adapted to prove the lower bound $\Omega(\varepsilon^{-1}d\log n - \varepsilon^{-1}\log\varepsilon^{-1})$ for ε-testing monotonicity on the hypergrid $[n]^d$. This coincides with the bound by Chakrabarty and Seshadhri [5], but, again, our construction bypasses Ramsey theory, which results in a vastly reduced range size.

2 Preliminaries

Although this is not standard, it will be convenient for us to denote $[n] = \{0, 1, \ldots, n-1\}$, and $[a..b] = \{a, a+1, \ldots, b-1\}$. Thus, $[n] = [0..n]$, and note that $[a..b]$ does not contain b.

An *assignment* $\alpha\colon S \to [r]$ is a function defined on a subset $S \subseteq [n]$. The *weight* of α is the size of S. We say that f *agrees* with α if $f(x) = \alpha(x)$ for all $x \in S$. This is notated by $f \rightharpoonup \alpha$. The choice of notation is to distinguish it from $f \sim \mu$ which means that f is distributed according to the probability distribution μ.

All logarithms are to the base of 2.

3 Proof of Theorem 1

We will define a probability distribution μ on monotone functions $f\colon [2^k] \to [k^{3k}]$ and a probability distribution ν on functions $g\colon [2^k] \to [k^{3k}]$ that are 1/2-far from monotone. It will be impossible to distinguish these two distributions using fewer than $\Omega(k)$ queries. Let $m = k^3$, so that $r = m^k$.

We define the distribution μ in two different but equivalent ways. First, μ is defined as the last member in an inductively-defined family $\mu_0, \mu_1, \ldots, \mu_k$ of distributions, where μ_i is supported on functions $[2^i] \to [m^i]$. The distribution μ_0 is supported on the only function that maps 0 to 0. Assume that μ_i is already defined and let us define μ_{i+1}. In order to do that, we independently sample f_0 and f_1 from μ_i and a from $[m-1]$. The corresponding function f in μ_{i+1} is given by

$$f(x) = \begin{cases} a \cdot m^i + f_0(x), & \text{if } 0 \le x < 2^i; \\ (a+1)m^i + f_1(x - 2^i), & \text{if } 2^i \le x < 2^{i+1}. \end{cases} \tag{1}$$

For an alternative way of defining μ, let us assume that the argument x is written in binary and the value $f(x)$ in m-ary. We prepend leading zeroes if necessary so that each number has exactly k digits. We enumerate the digits from left to right with the elements of $[k]$, so that the 0-th digit is the most significant one, and the $(k-1)$-st digit is the least significant one. For each binary string s of length strictly less than k, sample an element a_s from $[m-1]$ independently and uniformly at random. The i-th digit of $f(x)$ is defined as $a_s + b$, where s is the prefix of x of length i and b is the i-th bit of x. It is easy to see that both definitions of μ are equivalent, and that any function f from the support of μ is monotone.

The distribution ν is defined as the uniform mixture of the following distributions ν^j for $j \in [k]$. The function $g \sim \nu^j$ is defined as $g(x) = f(x \oplus 2^{k-1-j})$ when f is sampled from μ. Here $\oplus$ denotes the bit-wise XOR function. In other words, the jth bit of the argument is flipped before applying f. Alternatively, we may say that $g \sim \nu^j$ is defined as in the case of μ with the exception that the j-th digit of $g(x)$ is $a_s + (1-b)$ instead of $a_s + b$.

▶ **Claim 2.** *Any function g in the support of ν^j is 1/2-far from monotone.*

Proof. Consider two input strings $x < y$ that differ only in the j-th bit. By the definition of ν^j we have $g(x) > g(y)$, thus, $\{x, y\}$ is a monotonicity-violating pair. We have 2^{k-1} such disjoint pairs, and every monotone function differs from g on at least one element of each pair. ◀

Now we are ready to start with the proof of Theorem 1. Assume towards contradiction that there exists a randomised 1/2-tester $\mathcal{A}$ with query complexity $o(k)$. Using standard error reduction, we may assume that $\mathcal{A}$ errs with probability at most 1/8 on each input. Let λ be the uniform mixture of μ and ν. Clearly, $\mathcal{A}$ errs with probability at most 1/8 on λ. The randomised algorithm $\mathcal{A}$ can be defined as a probability distribution on deterministic query algorithms of the same query complexity, hence, one of the deterministic algorithms in the support of $\mathcal{A}$ errs with probability at most 1/8 on λ. Thus, we may assume $\mathcal{A}$ is a *deterministic* query algorithm. The error probability 1/8 on λ implies that on μ and ν we have

$$\Pr_{f\sim\mu}[\mathcal{A} \text{ accepts } f] \geq \frac{3}{4} \qquad \text{and} \qquad \Pr_{g\sim\nu}[\mathcal{A} \text{ accepts } g] \leq \frac{1}{4}. \tag{2}$$

Also, we may assume that k is large enough, so that the query complexity of the deterministic query algorithm $\mathcal{A}$ is less than $k/2$.

So, $\mathcal{A}$ is a decision tree. Its leaves are specified by assignments in the sense that $\mathcal{A}$ terminates its work on input f in a leaf given by assignment α if and only if f agrees with α. Moreover, the weight of each such α does not exceed the query complexity of $\mathcal{A}$. We partition all these assignments α into two parts as follows. We say that an integer $\ell \in [r]$ is *good* if it contains no digit 0 and no digit $m-1$ (in the m-ary representation as before). Otherwise, ℓ is *bad*. We say that an assignment α is *good* if all the elements in its image are good. Otherwise, α is *bad*. Finally, a leaf of $\mathcal{A}$ is good iff the corresponding assignment is good.

▶ **Claim 3.** *The probability that $\mathcal{A}$ ends its work in a bad leaf when run on an input f sampled from μ is $o(1)$.*

Proof. For each x in the domain of f, the value $f(x)$ is bad only if one of the k elements $a_{s_0}, a_{s_1}, \ldots, a_{s_{k-1}}$ has value 0 or $m-2$, where s_i is the prefix of x of length i. Thus, the probability of $\mathcal{A}$ to terminate in a bad leaf is at most the probability of finding an element a_s with value in $\{0, m-2\}$ with $k/2$ queries, where on each query it is allowed to test the values of k different a_s's. Since the a_s's are independent, and the probability of $a_s \in \{0, m-2\}$ is $2/(m-1)$, we get, using the standard bound on search, that the probability of succeeding is at most $\frac{k}{2} \cdot k \cdot \frac{2}{m-1} = o(1)$. ◀

▶ **Claim 4.** *For each good assignment α of weight at most $k/2$,*

$$\Pr_{f\sim\mu}[f \rightarrowtail \alpha] \leq 2 \cdot \Pr_{g\sim\nu}[g \rightarrowtail \alpha].$$

Before we start with the proof of this claim, let us show how Theorem 1 follows from Claims 3 and 4. In the following, let C be the set of assignments which correspond to the accepting leaves of $\mathcal{A}$, let $B \subseteq C$ be the subset of bad assignments, and $G \subseteq C$ be the subset of good assignments. Then,

$$\begin{aligned}\Pr_{f\sim\mu}[\mathcal{A} \text{ accepts } f] &= \sum_{\alpha\in B} \Pr_{f\sim\mu}[f \rightarrowtail \alpha] + \sum_{\alpha\in G} \Pr_{f\sim\mu}[f \rightarrowtail \alpha] \\ &\leq o(1) + 2\sum_{\alpha\in G} \Pr_{g\sim\nu}[g \rightarrowtail \alpha] \leq o(1) + 2\Pr_{g\sim\nu}[\mathcal{A} \text{ accepts } g],\end{aligned}$$

which is in contradiction with (2) if k is large enough.

It remains to prove good. Consider a good assignment α of weight at most $k/2$. Let S be the domain of α. We say that a pair $x < y$ from S *cuts* an index $j \in [k]$ iff their first j bits agree, and they disagree in the j-th bit. In other words, there exists $a \in \mathbb{Z}$ such that

$$2a \cdot 2^{k-j-1} \leq x < (2a+1) \cdot 2^{k-j-1} \leq y < (2a+2) \cdot 2^{k-j-1}.$$

The assignment α cuts all the indices cut by the pairs in S. good follows from the following two lemmata.

▶ **Lemma 5.** *If a good assignment α does not cut an index j, then*

$$\Pr_{f \sim \mu}[f \rightarrow \alpha] = \Pr_{g \sim \nu^j}[g \rightarrow \alpha].$$

Proof. By induction on i in the definition (1) of μ_i. Let for brevity $j' = k - j - 1$. If $j' < i$, we define ν_i^j as the distribution over the functions $g(x) = f(x \oplus 2^{j'})$ when f is sampled from μ_i. If $j' \geq i$, we define $\nu_i^j = \mu_i$. We prove that

$$\Pr_{f \sim \mu_i}[f \rightarrow \alpha] = \Pr_{g \sim \nu_i^j}[g \rightarrow \alpha] \tag{3}$$

for every good assignment α from $[2^i]$ to $[m^i]$ that does not cut the index j.

The base case $i = 0$ is trivial. (Actually, the statement is trivial for all $i \leq j'$.) Assume (3) is proven for i, and let us prove it for $i+1$. Let S be the domain of α. There are two cases.

First, assume both $S \cap [2^i]$ and $S \cap [2^i..2^{i+1}]$ are non-empty. This means that α cuts $k - i - 1$, hence, $i \neq j'$. Also, we may assume there exists $1 \leq a \leq m-3$ such that $\alpha([2^i]) \subseteq [am^i..(a+1)m^i]$ and $\alpha([2^i..2^{i+1}]) \subseteq [(a+1)m^i..(a+2)m^i]$, since otherwise both sides of (3) are 0. Under these assumptions,

$$\Pr_{f \sim \mu_{i+1}}[f \rightarrow \alpha] = \frac{1}{m-1} \Pr_{f_0 \sim \mu_i}[f_0 \rightarrow \alpha_0] \Pr_{f_1 \sim \mu_i}[f_1 \rightarrow \alpha_1],$$

where f_0 and f_1 are obtained reversely from (1), and α_0 and α_1 are defined similarly: $\alpha_0(i) = \alpha(i) \bmod m^i$ and $\alpha_1(i) = \alpha(i + 2^i) \bmod m^i$ for all $i \in [2^i]$. Both assignments are good, and they do not cut the index j. Similarly, since $i \neq j'$:

$$\Pr_{g \sim \nu_{i+1}^j}[g \rightarrow \alpha] = \frac{1}{m-1} \Pr_{g_0 \sim \nu_i^j}[g_0 \rightarrow \alpha_0] \Pr_{g_1 \sim \nu_i^j}[g_1 \rightarrow \alpha_1].$$

By the inductive assumption, we have the required equality.

Now assume one of $S \cap [2^i]$ and $S \cap [2^i..2^{i+1}]$ is empty. We consider the case $S \subseteq [2^i]$, the second one being similar. Again, we can assume there exists $1 \leq a \leq m-2$ such that $\alpha([2^i]) \subseteq [am^i..(a+1)m^i]$. Then,

$$\Pr_{f \sim \mu_{i+1}}[f \rightarrow \alpha] = \frac{1}{m-1} \Pr_{f_0 \sim \mu_i}[f_0 \rightarrow \alpha_0]$$

and, no matter whether $i = j'$ or not,

$$\Pr_{g \sim \nu_{i+1}^j}[g \rightarrow \alpha] = \frac{1}{m-1} \Pr_{g_0 \sim \nu_i^j}[g_0 \rightarrow \alpha_0],$$

and again we have the required equality by the inductive assumption. ◀

▶ **Lemma 6.** *An assignment α of weight t cuts at most $t-1$ indices in $[k]$.*

Proof. Let S be the domain of α, and let $J \subseteq [k]$ be the set of indices that α cuts.

Let us construct a graph G as follows. Its vertex set is S. For every index $j \in J$ take one arbitrary pair of elements $x, y \in S$ that cuts j and connect x and y by an edge. We say that the edge xy cuts j.

We claim that the graph G is acyclic, from which the statement of the lemma follows. Assume that G contains a simple cycle. Consider an edge xy that cuts the minimal index j on this cycle. Then x and y disagree in the j-th bit. On the other hand, considering the remaining part of the cycle, we see that x and y agree in the j-th bit. A contradiction, hence, G is acyclic. ◀

By cut, there are at least $k/2$ indices not cut by α, and using goodalpha, we have

$$2 \cdot \Pr_{g\sim\nu}[g \rightharpoonup \alpha] \geq \frac{2}{k} \sum_{j:\alpha \text{ does not cut } j} \Pr_{g\sim\nu^j}[g \rightharpoonup \alpha] \geq \frac{2}{k} \cdot \frac{k}{2} \cdot \Pr_{f\sim\mu}[f \rightharpoonup \alpha] = \Pr_{f\sim\mu}[f \rightharpoonup \alpha],$$

proving good.

4 The Case of Small ε

In this section, we briefly describe how the result of Theorem 1 can be extended to arbitrary values of ε, and give an improved version of the algorithm by Ergün et al. [12].

▶ **Theorem 7.** *If $\varepsilon \leq 1/2$, the complexity of ε-testing a function $f\colon [n] \to [r]$ for monotonicity is*

$$\Omega\left(\min\left\{\frac{\log(\varepsilon n)}{\varepsilon}, \frac{\log(\varepsilon r)}{\varepsilon \log\log(\varepsilon r)}\right\}\right).$$

Proof. The proof closely follows that of Theorem 1. We will briefly describe the construction and the proof using the notation of Section 3.

Let ℓ be a positive integer and assume $\varepsilon = 1/(2\ell)$. We will construct probability distributions $\widetilde{\mu}$ and $\widetilde{\nu}$ on functions $f\colon [\ell 2^k] \to [\ell k^{3k}]$ such that all functions in the support of $\widetilde{\mu}$ are monotone, functions in the support of $\widetilde{\nu}$ are ε-far from monotone, and it takes $\Omega(\ell k)$ queries to distinguish $\widetilde{\mu}$ and $\widetilde{\nu}$. Expressing ℓk in terms of ε and n or in terms of ε and r gives the required bound.

Let μ and ν be as in Section 3. For $s \in [\ell]$, independently sample f_s from μ. Define $f \sim \widetilde{\mu}$ as

$$f(s \cdot 2^k + x) = s \cdot k^{3k} + f_s(x) \tag{4}$$

for all $s \in [\ell]$ and $x \in [2^k]$. The distribution $\widetilde{\nu}$ is defined as the uniform mixture of $\widetilde{\nu}^{t,j}$ as t ranges over $[\ell]$ and j over $[k]$. The corresponding function $f \sim \widetilde{\nu}^{t,j}$ is defined as in (4) with exception that f_t is sampled from ν^j instead of μ. It is easy to see that functions in the support of $\widetilde{\mu}$ are monotone, and, using non-monotone, that functions in the support of $\widetilde{\nu}$ are ε-far from monotone.

Informally, it takes $\Omega(\ell k)$ queries to distinguish $\widetilde{\mu}$ and $\widetilde{\nu}$ because we are searching for one non-monotone distribution ν^j among ℓ independent distributions. Formally, we may proceed as follows. Assume towards contradiction that there exists a deterministic query algorithm $\mathcal{A}$ that makes less than $\ell k/4$ queries, accepts $\widetilde{\mu}$ with probability at least $3/4$ and accepts $\widetilde{\nu}$ with probability at most $1/4$.

Using the same reasoning as in bad, the expected number of bad elements found by $\mathcal{A}$ when run on $\widetilde{\mu}$ is $O(\ell k^2/m) = o(\ell)$. We call an assignment α on $[\ell 2^k]$ *bad* if it has more than $\ell/4$ bad elements in its image. Otherwise, we call α good. By Markov's inequality, the probability $\mathcal{A}$ terminates in a bad assignment when executed on $\widetilde{\mu}$ is $o(1)$.

Now consider a good assignment α of weight at most $\ell k/4$. It corresponds to ℓ sub-assignments α_s on $[2^k]$ defined by $\alpha_s(x) = \alpha(s \cdot 2^k + x) \bmod k^{3k}$. Using good, we get that

$$\Pr_{f\sim\widetilde{\mu}}[f \rightharpoonup \alpha] = \Pr_{g\sim\widetilde{\nu}^{t,j}}[g \rightharpoonup \alpha] \tag{5}$$

if the sub-assignment α_t is good and does not cut j. Using that there are at most $\ell/4$ bad α_s and cut, we have that there are at least $\ell k/2$ pairs (t, j) satisfying (5). Hence,

$$\Pr_{f\sim\widetilde{\mu}}[f \rightharpoonup \alpha] \leq 2 \cdot \Pr_{g\sim\nu}[g \rightharpoonup \alpha].$$

Now we finish the proof as in Section 3. ◀

▶ **Theorem 8.** *Assume $\varepsilon n \geq 2$. Then, there exists a non-adaptive 1-sided algorithm that tests a function $f\colon [n] \to [r]$ for monotonicity using $O\big(\frac{\log(\varepsilon n)}{\varepsilon}\big)$ queries.*

Combined with the result of Theorem 7, we get that complexity of this problem is $\Theta\big(\frac{\log(\varepsilon n)}{\varepsilon}\big)$ if $\varepsilon n \geq 2$, and $\Theta(n)$ otherwise.

Proof. The algorithm is inspired by that of Ergün et al. [12]. The main difference is that the i in the loop in line 1c only goes to $\log(\varepsilon n)$ instead of $\log n$.

1. Repeat $\Theta(1/\varepsilon)$ times:
   a. Choose $x \in [n]$ uniformly at random.
   b. Query $f(x)$.
   c. For $i = 0, \ldots, \lceil \log(\varepsilon n) \rceil$:
      - Let w be the largest multiple of 2^i strictly smaller than x, and y be the smallest multiple of 2^i strictly larger than x. (If any of them is outside $[n]$, do not use it on the next steps.)
      - Query $f(w)$ and $f(y)$.
      - If $f(w) > f(x)$ or $f(x) > f(y)$, reject the function f.
2. If no contradiction to monotonicity was found, accept.

Clearly, the query complexity of the algorithm is $O\big(\frac{\log(\varepsilon n)}{\varepsilon}\big)$, it is non-adaptive, and it always accepts a monotone function f. Assume now that f is ε-far from monotone.

▶ **Lemma 9.** *If f is ε-far from monotone, there exists a collection of pairwise disjoint pairs $(x_1, y_1), \ldots, (x_t, y_t)$ for $t = \lfloor \varepsilon n/2 \rfloor$ such that, for all i, $x_i < y_i$, $f(x_i) > f(y_i)$, and $y_i - x_i \leq \varepsilon n$.*

Proof. The pairs can be constructed using the following algorithmic procedure. Start with $S \leftarrow [n]$ and $i \leftarrow 1$. We treat S as a sorted list. While $i \leq t$, choose two neighbouring elements $x_i < y_i$ in S such that $f(x_i) > f(y_i)$, remove x_i and y_i from S, and increment i.

It remains to prove that (a) such a pair (x_i, y_i) will always exist and (b) that $y_i - x_i \leq \varepsilon n$. For (a), observe that $i \leq t$ implies that we have removed strictly less than εn elements from

S so far. Since f is ε-far from monotone, we have that f restricted to S is not monotone (otherwise, it would be possible to extend $f|_S$ to a monotone function on all $[n]$). The existence of the pair (x_i, y_i) is now obvious.

For (b), again, observe that we have removed less than $\varepsilon n - 1$ elements from S so far. The elements x_i and y_i are neighbouring in S, which means they are at distance at most εn in the original list $[n]$. ◀

▶ **Lemma 10.** *Assume x and y satisfy $x < y$, $f(x) > f(y)$ and $y - x \leq \varepsilon n$. Then one element $z \in \{x, y\}$ of these two is such that the algorithm will reject if it chooses z on step 1(a).*

Proof. If $y = x + 1$, the algorithm will reject if it chooses either of x or y. So, assume $y \geq x + 2$.

We claim that there exists an integer $0 \leq i \leq \lceil \log(\varepsilon n) \rceil$ such that there is unique multiple of 2^i strictly between x and y. Indeed, there is at least one for $i = 0$ and at most one for $i = \lceil \log(\varepsilon n) \rceil$. Also, it is not possible that there is more than one for some value of i and zero for $i + 1$.

Let w be this unique multiple of 2^i. If $f(w) < f(x)$, then it will be detected if the algorithm chooses x on step 1(a). If $f(w) \geq f(x)$, then $f(w) > f(y)$, and it will be detected if the algorithm chooses y on step 1(a). ◀

By the previous two lemmata, there exist $\lfloor \varepsilon n/2 \rfloor$ values of x on which the algorithm rejects should it choose any of them on step 1(a). The probability of choosing one of them on one iteration of the loop is $\Omega(\varepsilon)$. The probability to choose any of them on one of the $\Theta(1/\varepsilon)$ iterations of the loop is $\Omega(1)$. ◀

5 The Case of Hypergrids

In this section, we show how our results can be transformed into a lower bound for testing monotonicity on the hypergrid.

▶ **Theorem 11.** *If $\varepsilon \leq 1/2$, the complexity of ε-testing a function $f\colon [n]^d \to [r]$ for monotonicity is*

$$\Omega\left(\min\left\{\frac{\log(\varepsilon n^d)}{\varepsilon}, \frac{\log(\varepsilon r)}{\varepsilon \log\log(\varepsilon r)}\right\}\right).$$

In terms of n, the lower bound can be expressed as $\Omega(\varepsilon^{-1} d \log n - \varepsilon^{-1} \log \varepsilon^{-1})$.

Proof. Consider the functions defined in the proof of Theorem 7. Assume that $\ell = 2^a$ for some integer a, and choose k so that $a + k = db$ for some integer b. We consider the domain of the input functions $[\ell 2^k] = [2^{a+k}]$ as $[2^b]^d$, where we break the binary representation of $x \in [2^{a+k}]$ into d groups of b bits.

If a function is monotone on $[2^{a+k}]$, it is still monotone when considered on $[2^b]^d$. Also, the analysis of the algorithm does not involve the order relation defined on the domain of the input functions. The only thing that might go wrong is that the functions in the support of $\widetilde{\nu}$ are no longer ε-far from monotone when considered on $[2^b]^d$. But this does not happen. Indeed, in the proof of non-monotone, the monotonicity-violating pairs (x, y) differ in exactly one bit. So if $x < y$ in $[2^{a+k}]$, this is still true in $[2^b]^d$, hence the proof of non-monotone carries over. Thus, all the functions in the support of $\widetilde{\nu}$ are ε-far from monotone. The statement of the theorem now follows from the proof of Theorem 7. ◀

References

1 Aleksandrs Belovs and Eric Blais. Quantum algorithm for monotonicity testing on the hypercube. *Theory of Computing*, 11(16):403–412, 2015.

2 Aleksandrs Belovs and Eric Blais. A polynomial lower bound for testing monotonicity. In *Proc. of 48th ACM STOC*, pages 1021–1032, 2016.

3 Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. *Computational Complexity*, 21(2):311–358, 2012.

4 Eric Blais, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Lower bounds for testing properties of functions over hypergrid domains. In *Proc. of 29th IEEE CCC*, pages 309–320, 2014.

5 Deeparnab Chakrabarty and C. Seshadhri. An optimal lower bound for monotonicity testing over hypergrids. *Theory of Computing*, 10:453–464, 2014.

6 Deeparnab Chakrabarty and Comandur Seshadhri. A $o(n)$ monotonicity tester for Boolean functions over the hypercube. In *Proc. of 45th ACM STOC*, pages 411–418, 2013.

7 Deeparnab Chakrabarty and Comandur Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Proc. of 45th ACM STOC*, pages 419–428, 2013.

8 Xi Chen, Anindya De, Rocco A. Servedio, and Li-Yang Tan. Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proc. of 47th ACM STOC*, pages 519–528, 2015.

9 Xi Chen, Rocco A. Servedio, and Li-Yang Tan. New algorithms and lower bounds for monotonicity testing. In *Proc. of 55th IEEE FOCS*, pages 286–295, 2014.

10 Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond Talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proc. of 49th ACM STOC*, pages 523–536, 2017.

11 Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *Proc. of 3rd*, pages 97–108. Springer, 1999.

12 Funda" Ergün, Sampath" "Kannan, S Ravi" "Kumar, Ronitt" "Rubinfeld, and Mahesh "Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60(3):717–751, 2000.

13 Eldar Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004.

14 Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In *Proc. of 34th ACM STOC*, pages 474–483, 2002.

15 Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

16 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

17 Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and Boolean isoperimetric type theorems. In *Proc. of 56th IEEE FOCS*, pages 52–58, 2015.

18 Ramesh Krishnan S Pallavoor, Sofya Raskhodnikova, and Nithin Varma. Parameterized property testing of functions. In *Proc. of 8th ACM ITCS*, volume 67 of *LIPIcs*, page 12. Dagstuhl, 2017.

19 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.