L_p-Testing

Piotr Berman
Pennsylvania State University
berman@cse.psu.edu

Sofya Raskhodnikova*
Pennsylvania State University
and Boston University
sofya@cse.psu.edu

Grigory Yaroslavtsev†
ICERM, Brown University
grigory@grigory.us

ABSTRACT

We initiate a systematic study of sublinear algorithms for approximately testing properties of real-valued data with respect to L_p distances for $p = 1, 2$. Such algorithms distinguish datasets which either have (or are close to having) a certain property from datasets which are far from having it with respect to L_p distance. For applications involving noisy real-valued data, using L_p distances allows algorithms to withstand noise of bounded L_p norm. While the classical property testing framework developed with respect to Hamming distance has been studied extensively, testing with respect to L_p distances has received little attention.

We use our framework to design simple and fast algorithms for classic problems, such as testing monotonicity, convexity and the Lipschitz property, and also distance approximation to monotonicity. In particular, for functions over the hypergrid domains $[n]^d$, the complexity of our algorithms for all these properties does not depend on the linear dimension n. This is impossible in the standard model. Most of our algorithms require minimal assumptions on the choice of sampled data: either uniform or easily samplable random queries suffice. We also show connections between the L_p-testing model and the standard framework of property testing with respect to Hamming distance. Some of our results improve existing bounds for Hamming distance.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]; F.1.1 [**Theory of Computation**]: Models of Computation—*Relations Between Models*

*This author was supported by NSF CAREER award CCF-0845701 and the Hariri Institute for Computing and Computational Science and Engineering at Boston University.

†This author was supported by NSF CAREER award CCF-0845701, a College of Engineering Fellowship at Penn State, and the Institute Postdoctoral Fellowship in Mathematics at the Institute for Computational and Experimental Research in Mathematics, Brown University.

STOC '14, May 31 - June 03 2014, New York, NY USA

Keywords

Property testing; monotone, Lipschitz and submodular functions; approximating distance to a property.

1. INTRODUCTION

Property testing [26, 18] is a rigorous framework for approximate analysis of global properties of data, which can be performed given access to a small sample. For example, one can approximately verify whether a (possibly multidimensional) array of numbers is sorted by examining a carefully chosen subset of its elements [11, 19, 9]. Formally, a property testing problem can be stated by treating data as a function on the underlying domain. The datasets satisfying a property form a class of functions. E.g., the set of all sorted real-valued n-element datasets corresponds to the class of monotone functions $f\colon [n] \to \mathbb{R}$, while monotone functions $f\colon [n]^d \to \mathbb{R}$ represent d-dimensional arrays with linear size n, sorted in each of the d dimensions[1]. The problem of testing sortedness can be formalized as follows. Let $\mathcal{M}$ be the class of all monotone functions. Given a function $f\colon [n]^d \to \mathbb{R}$ and a proximity parameter ε, we want to decide whether $f \in \mathcal{M}$ or f is at distance at least ε from any function in $\mathcal{M}$. The distance measure in the standard model is (relative) Hamming distance.

In this paper, we initiate a systematic study of properties of functions with respect to L_p distances, where $p > 0$. Property testing was originally introduced for analysis of algebraic properties of functions over finite fields such as linearity and low degree. Attention to these properties was motivated by applications to Probabilistically Checkable Proofs. In such applications, Hamming distance between two functions is a natural choice because of its interpretation as the probability that the two functions differ on a random point from the domain. Also, many initial results in property testing focused on Boolean functions, for which L_p distances are the same for $p = 0, 1$ and, more generally, are related in a simple way for different values of p, so all choices of p lead to the same testing problems. Subsequently, property testing algorithms have been developed for multiple basic properties of functions over the reals (e.g., monotonicity, convexity, submodularity, the Lipschitz property, etc.). We study testing these properties w.r.t. L_p distances, providing different approximation guarantees, better suited for many applications with real-valued data.

L_p-testing. Let f be a real-valued function over a fi-

[1]We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

nite[2] domain D. For $p \geq 1$, the L_p-norm of f is $\|f\|_p = \left(\sum_{x \in D} |f(x)|^p\right)^{1/p}$. For $p = 0$, let $\|f\|_0 = \sum_{x \in D} \|f(x)\|^0$ be the number of non-zero values of f. Let $\mathbb{1}$ denote the function that evaluates to 1 on all $x \in D$. A *property* $\mathcal{P}$ is a set of functions over D. For real-valued functions $f : D \to [0,1]$ and a property $\mathcal{P}$, we define relative L_p distance as follows[3]:

$$d_p(f, \mathcal{P}) = \inf_{g \in \mathcal{P}} \frac{\|f - g\|_p}{\|\mathbb{1}\|_p} = \inf_{g \in \mathcal{P}} \left(\mathbb{E}[|f - g|^p]\right)^{1/p},$$

where the first equality holds for $p \geq 0$ and the second for $p > 0$. The normalization by a factor $\|\mathbb{1}\|_p$ ensures that $d_p(f, \mathcal{P}) \in [0,1]$. For $p \geq 0$, a function f is ε-far from a property $\mathcal{P}$ w.r.t. the L_p distance if $d_p(f, \mathcal{P}) \geq \varepsilon$. Otherwise, f is ε-close to $\mathcal{P}$.

DEFINITION 1.1. *An L_p-tester for a property $\mathcal{P}$ is a randomized algorithm that, given a proximity parameter $\varepsilon \in (0,1)$ and oracle access to a function $f : \mathcal{D} \to [0,1]$,*

1. *accepts with probability at least 2/3 if $f \in \mathcal{P}$;*
2. *rejects with probability at least 2/3 if f is ε-far from $\mathcal{P}$ w.r.t. the L_p distance.*

The corresponding algorithmic problem is called *L_p-testing*. Standard property testing corresponds to L_0-testing, which we also call *Hamming testing*.

Tolerant L_p-testing and L_p-distance approximation. An important motivation for measuring distances to properties of real-valued functions w.r.t. L_p metrics is noise-tolerance. In order to be able to withstand noise of bounded Hamming weight (small number of outliers) in the property testing framework, Parnas, Ron, and Rubinfeld [23] introduced *tolerant* property testing. One justification for L_p-testing is that in applications involving real-valued data, noise added to the function often has large Hamming weight, but bounded L_p-norm for some $p > 0$ (e.g., Brownian motion, white Gaussian noise, etc.). This leads us to the following definition, which generalizes tolerant testing [23].

DEFINITION 1.2. *An $(\varepsilon_1, \varepsilon_2)$-tolerant L_p-tester for a property $\mathcal{P}$ is a randomized algorithm which, given $\varepsilon_1, \varepsilon_2 \in (0,1)$, where $\varepsilon_1 < \varepsilon_2$, and oracle access to a function $f : \mathcal{D} \to [0,1]$,*

1. *accepts with probability at least 2/3 if f is ε_1-close to $\mathcal{P}$ with respect to L_p distance.*
2. *rejects with probability at least 2/3 f is ε_2-far from $\mathcal{P}$ with respect to L_p distance.*

For example, a tolerant L_1-tester can ignore both uniform noise of bounded magnitude and noise of large magnitude concentrated on a small set of outliers.

[2]Some of our results apply to functions over infinite measurable domains, i.e., $\int_{\mathcal{D}} \mathbb{1} < \infty$, where $\mathbb{1}$ is an indicator function of the domain $\mathcal{D}$. This is why we use the name L_p rather than ℓ_p. An important example of such a domain is the hypercube $[0,1]^d$ in $\mathbb{R}^d$.

[3]The definition of distance d_p can be extended to functions $f : D \to [a,b]$, where $a < b$, by changing the normalization factor to $\|\mathbb{1}\|_p \cdot (b-a)$. Our results hold for the more general range (if the algorithms are given bounds a and b), since for the properties we consider (monotonicity, the Lipschitz property and convexity), testing f reduces to testing $f' = \frac{f(x)-a}{b-a}$. For ease of presentation, we set the range to $[0,1]$.

A related computational task is *approximating the L_p distance to a property $\mathcal{P}$:* given oracle access to a function f, output an additive approximation to the distance $d_p(f, \mathcal{P})$, which has the desired accuracy with probability at least 2/3. Distance approximation is equivalent to tolerant testing (up to small multiplicative factors in the running time) [23]. Both problems were studied extensively for monotonicity [23, 1, 27, 13] and convexity [12]. Despite significant progress, an optimal algorithm is not known even for monotonicity in one dimension. In contrast, for L_1-testing we are able to fully resolve this question for one-dimensional functions.

Connections with learning. Another compelling motivation for L_p-testing comes from learning theory. It has been pointed out that property testing can be helpful in model selection. If the concept class for learning a target function is not known reliably in advance, one can first run more efficient property testing or distance approximation algorithms in order to check multiple candidate concept classes for the subsequent learning step. It is important that the approximation guarantees of the preprocessing step and the learning step be aligned. Because L_p distances are frequently used to measure error in PAC-learning of real-valued functions (see e.g. [20]), an L_p-testing algorithm is a natural fit for preprocessing in such applications, especially when noisy real-valued data is involved. We believe that investigating L_p-testing might be an important step in bridging the gap between existing property testing and learning models.

We also note that the well established connection between Hamming testing and learning [18] naturally extends to L_p-testing, and we exploit it in our results on monotonicity and convexity (see Section 1.2). Namely, from the information-theoretic perspective, property testing is not harder than PAC-learning (up to a small additive factor), although computationally this holds only for proper learning. Tolerant testing and distance approximation are related in a similar way to agnostic learning [23]. Thus, the goal of property testing is to design algorithms which have significantly lower complexity than the corresponding learning algorithms and even go beyond the lower bounds for learning.

Connections with approximation theory. Another closely related field is that of approximation theory. Computational tasks considered in that field (e.g., approximating a class of functions with L_p error) are similar to learning tasks. Basically, the only differences between approximation and learning tasks are that approximation algorithms are usually allowed to query the input function at points of their choice and are non-agnostic (i.e., they work only under the assumption that the input function is in a given class). Approximation theory is not known to imply any interesting results for Hamming property testing primarily because approximation results usually have L_p error with $p > 0$. But once the L_p metric is considered in property testing, the parallels between the computational tasks studied in sublinear algorithms and approximation theory become apparent. In Section 1.2.3, we exploit the connection to approximation theory to get an L_p-testing algorithm for convexity.

Previous work related to L_p-testing. No prior work systematically studies L_p-testing of properties of functions. The only explicitly stated L_p-testing result for $p > 0$ is the L_1-tester for submodular functions in a recent paper by Feldman and Vondrák [14]. It is a direct corollary of their junta approximation result w.r.t. L_1 distance. The upper

bound on the query complexity of L_1-testing of submodularity in [14] is far from the known lower bound.

There are also property testing papers that work with L_1 distance, but consider different input access. Rademacher and Vempala [24] study property testing whether a given set S is convex, where ε-far means that there is no convex set K such that $vol(K \Delta S) \leq \varepsilon \cdot vol(S)$. In addition to oracle access, their algorithm can sample a random point form the set S. Finally, L_1 distance is also widely used in the line of work of testing properties of distributions started by [3].

1.1 Basic Relationships Between Models

Our first goal is to establish relationships between Hamming, L_1 and L_2-testing for standard and tolerant models[4].

L_p-testing. We denote the worst-case query complexity of L_p-testing for property $\mathcal{P}$ with proximity parameter ε by $Q_p(\mathcal{P}, \varepsilon)$. The following fact establishes basic relationships between L_p-testing problems and follows directly from the inequalities between L_p-norms. (See full version for details.)

FACT 1.1. *Let $\varepsilon \in (0,1)$ and $\mathcal{P}$ be a property over any domain. Then $Q_0(\mathcal{P}, \varepsilon) \geq Q_1(\mathcal{P}, \varepsilon) \geq Q_2(\mathcal{P}, \sqrt{\varepsilon}) \geq Q_1(\mathcal{P}, \sqrt{\varepsilon})$. Moreover, if $\mathcal{P}$ is a property of Boolean functions then $Q_0(\mathcal{P}, \varepsilon) = Q_1(\mathcal{P}, \varepsilon) = Q_2(\mathcal{P}, \sqrt{\varepsilon})$.*

This fact has several implications. First, L_1-testing is no harder than standard Hamming testing (the first inequality). Hence, upper bounds on the query complexity of the latter are the baseline for the design of L_1-testing algorithms. As we will demonstrate, for the properties considered in this paper, L_1-testing has significantly smaller query complexity than Hamming testing. This fact also shows that L_1 and L_2-testing problems are equivalent up to quadratic dependence on ε (the second and third inequalities). Finally, the equivalence of L_p-testing problems for Boolean functions implies that all lower bounds for such functions in the standard Hamming testing model are applicable to L_p-testing.

Tolerant L_p-testing. We denote the worst-case query complexity of $(\varepsilon_1, \varepsilon_2)$-tolerant L_p-testing of a property $\mathcal{P}$ by $Q_p(\mathcal{P}, \varepsilon_1, \varepsilon_2)$. Introducing tolerance complicates relationships between L_p-testing problems for different p as compared to the relationships in Fact 1.1. The proof of the following fact is deferred to the full version.

FACT 1.2. *Let $\varepsilon_1, \varepsilon_2 \in (0,1)$ such that $\varepsilon_1 < \varepsilon_2^2$ and $\mathcal{P}$ be a property over any domain. Then*

$$Q_1(\mathcal{P}, \varepsilon_1^2, \varepsilon_2) \leq Q_2(\mathcal{P}, \varepsilon_1, \varepsilon_2) \leq Q_1(\mathcal{P}, \varepsilon_1, \varepsilon_2^2).$$

Facts 1.1-1.2 establish the key role of L_1-testing in understanding property testing w.r.t. L_p distances since results for L_2-testing follow with a minor loss in parameters. Moreover, in many cases, these results turn out to be optimal.

1.2 Our Results

We consider three properties of real-valued functions: monotonicity, the Lipschitz property and convexity. We focus on understanding the L_1 distance to these properties and obtain results for L_2 distance by applying Facts 1.1-1.2. Most of our algorithms have additional guarantees, defined next.

[4]In the rest of the paper, we consider L_p-testing and distance approximation only for $p = 0, 1, 2$, leaving the remaining cases for future work.

DEFINITION 1.3. *An algorithm is called* nonadaptive *if it makes all queries in advance, before receiving any responses; otherwise, it is called* adaptive. *A testing algorithm for property $\mathcal{P}$ has* 1-sided error *if it always accepts all inputs in $\mathcal{P}$; otherwise, it has* 2-sided error.

1.2.1 Monotonicity

Monotonicity is perhaps the most investigated property in the context of property testing and distance approximation.

DEFINITION 1.4. *Let D be a (finite) domain equipped with a partial order $\preceq$. A function $f : D \to \mathbb{R}$ is* monotone *if $f(x) \leq f(y)$ for all $x, y \in D$ satisfying $x \preceq y$.*

An important specific domain D is a *d-dimensional hypergrid* $[n]^d$ equipped with the partial order $\preceq$, where $(x_1, \ldots, x_n) \preceq (y_1, \ldots, y_n)$ whenever $x_1 \leq y_1, \ldots, x_n \leq y_n$. The special case $[n]$ of the hypergrid is called a *line*, and the special case $[2]^d$ is a *hypercube*. These domains are interesting in their own right. For example, testing monotonicity on the line $[n]$ corresponds to testing whether a list of n numbers is sorted (in nondecreasing order). The L_1 distance to monotonicity on the line is the total change in the numbers required to make them sorted.

Characterization. We give a characterization of the L_1 distance to monotonicity in terms of the distance to monotonicity of Boolean functions (Lemma 2.1). The main idea in our characterization is that every function can be viewed as an integral over Boolean threshold functions. This view allows us to express the L_1-distance to monotonicity of a real-valued function $f\colon D \to [0,1]$ as an integral over the L_1-distances to monotonicity of its Boolean threshold functions. We use this characterization to obtain reductions from the general monotonicity testing and distance approximation to the case of Boolean functions (Section 2.1) that preserve query complexity and running time[5].

Recall that for Boolean functions, L_0 and L_1 distances are equal. Thus, our reductions allow us to capitalize on existing algorithms and lower bounds for (Hamming) testing of and distance approximation to monotonicity of Boolean functions. For example, for the case of the line domain $[n]$, it is folklore that monotonicity of Boolean functions can be tested nonadaptively and with 1-sided error in $O\left(\frac{1}{\varepsilon}\right)$ time. In contrast, for Hamming testing with the general range, one needs $\Omega\left(\frac{\log n}{\varepsilon} - \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ queries even for adaptive 2-sided error testers [11, 15, 6]. Therefore, testing monotonicity on the line is a factor of $\log n$ faster w.r.t. the L_1 distance than Hamming distance. A comparison between the query complexity of L_p-testing monotonicity on the line and the hypergrid for $p = 0, 1, 2$ is given in Table 1.1. Results for other domains are deferred to the full version of this paper.

[5]Our reductions are stated for nonadaptive algorithms. Such reductions are useful because all known upper bounds for testing monotonicity can be achieved by nondaptive testers, with one exception: our adaptive bound for testing Boolean functions on constant-dimensional hypergrids from Section 2.3. We can get a reduction that works for adaptive algorithms by viewing L_1-testing monotonicity as a multi-input concatenation problem [16]. This reduction preserves the query complexity for the special class of *proximity-oblivious* testers [8], but incurs a loss of $O\left(\frac{1}{\varepsilon}\right)$ in general and, specifically, when applied to our adaptive tester. As this approach would not improve our results, we focus on reductions for nonadaptive algorithms.

Monotonicity		
Domain	Hamming Testing	L_p-Testing for $p = 1, 2$
$[n]$	$O\left(\frac{\log n}{\varepsilon}\right)$ n.a. 1-s. [11]	$O\left(\frac{1}{\varepsilon^p}\right)$ n.a. 1-s. Lem. 2.2 + Fact 1.1
	$\Omega\left(\frac{\log n}{\varepsilon} - \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ a. 2-s. [11, 15, 6]	$\Omega\left(\frac{1}{\varepsilon^p}\right)$ a. 2-s. Fact 1.1
$[n]^d$	$O\left(\frac{d\log n}{\varepsilon}\right)$ n.a. 1-s. [5]	$O\left(\frac{d}{\varepsilon^p}\log\frac{d}{\varepsilon^p}\right)$ n.a. 1-s. Thm. 1.3 + Fact 1.1
	$\Omega\left(\frac{d\log n}{\varepsilon} - \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ a. 2-s. [6]	$\Omega\left(\frac{1}{\varepsilon^p}\log\frac{1}{\varepsilon^p}\right)$ n.a. 1-s. Thm. 2.12 + Fact 1.1

The c-Lipschitz property		
Domain	Hamming Testing	L_p-Testing for $p = 1, 2$
$[n]^d$	$O\left(\frac{d\log n}{\epsilon}\right)$ n.a. 1-s. [5]	$O\left(\frac{d}{\varepsilon^p}\right)$ n.a. 1-s. Thm. 1.4 + Fact 1.1
	$\Omega\left(\frac{d\log n}{\varepsilon} - \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ a. 2-s. [7]	$\Omega(d + \frac{1}{\varepsilon^p})$ a. 2-s. [21] + Fact 1.1

Table 1.1: Query complexity of L_p-testing monotonicity / the Lipschitz property of functions $f : \mathcal{D} \to [0, 1]$ for $p = 1, 2$ (a./n.a. = adaptive/nonadaptive, 1-s./2-s. = 1-sided error/2-sided error).

L_1-testing on hypergrids and Levin's work investment strategy. One of our reductions (described above) shows that the nonadaptive complexity of L_1-testing monotonicity is the same for functions over $[0, 1]$ and over $\{0, 1\}$. Dodis et al. [9] gave a monotonicity tester of Boolean functions on $[n]^d$ that makes $O\left(\frac{d}{\varepsilon}\log^2\frac{d}{\varepsilon}\right)$ queries and runs in $O\left(\frac{d}{\varepsilon}\log^3\frac{d}{\varepsilon}\right)$ time. We obtain a tester with better query and time complexity.

THEOREM 1.3. *Let $n, d \in \mathbb{N}$ and $\varepsilon \in (0, 1)$. The time complexity of L_1-testing monotonicity of functions $f : [n]^d \to [0, 1]$ with proximity parameter ε (nonadaptively and with one-sided error) is $O\left(\frac{d}{\varepsilon}\log\frac{d}{\varepsilon}\right)$.*

The test in [9] is based on the dimension reduction (stated as Theorem 2.4 in Section 2.2) and Levin's work investment strategy [22], described in detail by Goldreich [16]. Our improvement in the upper bound on the query complexity stems from an improvement to Levin's strategy. (The additional improvement in running time comes from a more efficient check for violations of monotonicity among sampled points.) As described in [16], Levin's strategy has been applied in many different settings [22], including testing connectedness of bounded-degree graphs in [17], testing connectedness of images in [25] and analyzing complexity of the concatenation problem [16]. Our improvement to Levin's strategy saves a logarithmic factor in the running time in these applications. Specifically, whether a graph of bounded degree is connected can be tested with $O\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ queries, whether an image represents a connected object can be tested with $O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)$ queries, and there is only an $O\left(\log\frac{1}{\varepsilon}\right)$ overhead in the query complexity of the concatenation of property testing instances as compared to solving a single instance.

Role of adaptivity. Researchers have repeatedly asked whether adaptivity is helpful in testing monotonicity. All previously known adaptive tests have been shown to have nonadaptive analogs with the same query and time complexity. (See, e.g., [4] for a discussion of both points.) Yet, for some domains and ranges there is a large gap between adaptive and nonadaptive lower bounds. We exhibit the first monotonicity testing problem where adaptivity provably helps: we show that for functions of the form $f : [n]^2 \to \{0, 1\}$, monotonicity testing can be performed with $O\left(\frac{1}{\varepsilon}\right)$ queries with an adaptive 1-sided error algorithm, while every nonadaptive 1-sided error algorithm for this task requires $\Omega\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ queries. Our upper bound of $O\left(\frac{1}{\varepsilon}\right)$ queries holds more generally: for any constant d. This upper bound is optimal because one needs $\Omega\left(\frac{1}{\varepsilon}\right)$ queries to test any nontrivial property, including monotonicity. Our lower bound shows that the tester from Theorem 1.3 is an optimal nonadaptive 1-sided error tester for hypergrids of constant dimension.

Our adaptive tester is based on an algorithm that partially learns the class of monotone Boolean functions over $[n]^d$. (The partial learning model is formalized in Definition 2.3. In particular, our partial learner implies a proper PAC learner under the uniform distribution with membership queries.) A straightforward transformation[6] gives a 1-sided error tester from the learner. For the special case of $d = 2$, the tester has the desired $O(\frac{1}{\varepsilon})$ query complexity. Our $O\left(\frac{1}{\varepsilon}\right)$-query tester for higher dimensions is more sophisticated: it uses our nonadaptive monotonicity tester (from Theorem 1.3) in conjunction with the learner. The idea is that the values previously deduced by the learner do not have to be queried, thus reducing the query complexity.

Our lower bound for nonadaptive testing is based on a hexagon packing.

Tolerant testing and distance approximation. In the full version of the paper, we give L_1-distance approximation algorithms with additive error δ for monotonicity of functions on the line and the 2-dimensional grid. The query complexity for the line and the grid are $O(1/\delta^2)$ and $\tilde{O}(1/\delta^4)$, respectively. Our algorithm for the line is optimal. It implies a tolerant L_1-tester for monotonicity on the line with query complexity $O\left(\frac{\varepsilon_2}{(\varepsilon_2-\varepsilon_1)^2}\right)$ and, by Fact 1.2, a tolerant L_2-tester for this problem with query complexity $O\left(\frac{\varepsilon_2^2}{(\varepsilon_2^2-\varepsilon_1)^2}\right)$.

A crucial building block of our algorithms is the reduction from the general approximation problem to the special case of Boolean functions, described above. For the line, we further reduce the problem to approximating the longest correct bracket subsequence. Our distance approximation algorithm for Boolean functions improves on the $\tilde{O}(1/\delta^2)$-query algorithm of Fattal and Ron [13]. For $d = 2$, we apply our reduction to the algorithm of [13] for Boolean functions.

1.2.2 The c-Lipschitz properties

The c-Lipschitz properties are a subject of a recent wave of investigation [21, 2, 8, 5, 7], with a focus on hypergrid domains, due to their applications to differential privacy [10].

[6] Our transformation can be viewed as an analog of Proposition 3.1 in [18]. This proposition relates the query complexity of 2-sided error testing to the sample complexity of proper learning. Our transformation requires a stronger learner and yields a 1-sided error tester.

DEFINITION 1.5. *Let (D, d_D) be a finite metric space, i.e., D is a finite set and $d_D : D \times D \to \mathbb{R}$ is a metric. Let* the Lipschitz constant $c > 0$ *be a real number. A function $f : D \to \mathbb{R}$ is* c-Lipschitz *if $|f(x) - f(y)| \leq c \cdot d_D(x, y)$ for all $x, y \in D$. If $c = 1$, such a function is called* Lipschitz.

The hypergrid, the hypercube and the line domains are defined as for monotonicity, except that instead of equipping them with the partial order, we equip them with the following metric: $d_D(x, y) = \|x - y\|_1$.

Characterization. We give a combinatorial characterization of the L_1 distance to the Lipschitz property (in Lemma 3.1). We show that it is equal to the weight of a maximum weight matching in the appropriately defined graph associated with the function. We note that a similar-looking near-characterization is known w.r.t. the Hamming distance, but the upper and lower bounds on the Hamming distance to the Lipschitz property are off by a factor of 2. Our characterization w.r.t. the L_1 distance is tight.

L_1-testing on hypergrids. We use our characterization to obtain a c-Lipschitz L_1-tester for functions over hypergrids that is faster by a factor of $\log n$ than the best possible Hamming tester. Known bounds on the query complexity of testing the c-Lipschitz property on hypergrids are summarized in Table 1.1.

THEOREM 1.4. *Let $n, d \in \mathbb{N}$ and $\varepsilon, c \in (0, 1)$. The time complexity of L_1-testing the c-Lipschitz property of functions $f : [n]^d \to [0, 1]$ (nonadaptively and with 1-sided error) with proximity parameter ε is $O\left(\frac{d}{\varepsilon}\right)$.*

The running time of our tester has optimal dependence on dimension d. This follows from the $\Omega(d)$ lower bound on Hamming testing of the Lipschitz property of functions $f : \{0, 1\}^d \to \{0, 1, 2\}$ in [21]. (This problem is equivalent to Hamming testing 1/2-Lipschitz property of functions $f : \{0, 1\}^d \to \{0, 1/2, 1\}$, and for functions with this range, relative L_0 and L_1 distances are off by at most factor of 2.)

The running time of our tester does not depend on the Lipschitz constant c, but the algorithm itself does. The crux of designing the algorithm is understanding the number of pairs of points on the same line which do not obey the Lipschitz condition (called *violated pairs*), and selecting the right subset of pairs (depending on c) so that a constant fraction of them are violated by any function on the line that is ε-far from monotone. The analysis uses dimension reduction from [2], generalized to work for functions with range $\mathbb{R}$.

1.2.3 Convexity and Submodularity

We establish and exploit the connection of L_1-testing to approximation theory. Our results for testing convexity of functions over $[n]^d$ (presented in the full version) follow from this connection. For $d = 1$, we get an optimal tester with query complexity $O(1/\epsilon)$ and for higher dimensions, query complexity is independent of the linear dimension n.

2. L_1-TESTING MONOTONICITY

2.1 Distance to Monotone: Characterization

We characterize the L_1 distance to monotonicity in terms of the distance to monotonicity of Boolean functions. We use this characterization to obtain reductions from the general monotonicity testing and distance approximation to the case of Boolean functions. The main idea in our characterization is that every function can be viewed as an integral over Boolean threshold functions, defined next.

DEFINITION 2.1. *For a function $f : D \to [0, 1]$ and $t \in [0, 1]$, the* threshold function $f_{(t)} : D \to \{0, 1\}$ *is:*

$$f_{(t)}(x) = \begin{cases} 1 & \text{if } f(x) \geq t; \\ 0 & \text{if } f(x) < t. \end{cases}$$

We can express a real-valued function $f : D \to [0, 1]$ as an integral over its Boolean threshold functions:

$$f(x) = \int_0^{f(x)} dt = \int_0^1 f_{(t)}(x) dt.$$

The integrals above and all other integrals in this section are well defined because we are integrating over piecewise constant functions.

Let $L_1(f, \mathcal{M})$ denote the L_1 distance from f to the set of monotone functions, $\mathcal{M}$, and let $d_{\mathcal{M}}(f)$ be the relative version of this distance, i.e., $d_{\mathcal{M}}(f) = L_1(f, \mathcal{M})/|D|$ for functions $f : D \to [0, 1]$.

LEMMA 2.1 (CHARACTERIZATION). *For every function $f : D \to [0, 1]$, the distance $d_{\mathcal{M}}(f) = \int_0^1 d_{\mathcal{M}}(f_{(t)})\, dt$.*

PROOF. Since f and $f_{(t)}$ are functions over the same domain, it is enough to prove $L_1(f, \mathcal{M}) = \int_0^1 L_1(f_{(t)}, \mathcal{M})\, dt$. First, we prove that $L_1(f, \mathcal{M}) \leq \int_0^1 L_1(f_{(t)}, \mathcal{M})\, dt$. For all $t \in [0, 1]$, let g_t be the closest monotone (Boolean) function to $f_{(t)}$. Define $g = \int_0^1 g_t\, dt$. Since g_t is monotone for all $t \in [0, 1]$, function g is also monotone. Then

$$\begin{aligned} L_1(f, \mathcal{M}) &\leq \|f - g\|_1 = \Big\| \int_0^1 f_{(t)}\, dt - \int_0^1 g_t\, dt \Big\|_1 \\ &= \Big\| \int_0^1 (f_{(t)} - g_t)\, dt \Big\|_1 \\ &\leq \int_0^1 \|f_{(t)} - g_t\|_1\, dt = \int_0^1 L_1(f_{(t)}, \mathcal{M})\, dt. \end{aligned}$$

Next, we prove that $L_1(f, \mathcal{M}) \geq \int_0^1 L_1(f_{(t)}, \mathcal{M})\, dt$. Let g denote the closest monotone function to f in L_1 distance. Then $g_{(t)}$ is monotone for all $t \in [0, 1]$. We obtain:

$$\begin{aligned} L_1(f, \mathcal{M}) &= \|f - g\|_1 = \Big\| \int_0^1 (f_{(t)} - g_{(t)})\, dt \Big\|_1 \\ &= \sum_{x : f(x) \geq g(x)} \int_0^1 (f_{(t)}(x) - g_{(t)}(x))\, dt \\ &\quad + \sum_{x : f(x) < g(x)} \int_0^1 (g_{(t)}(x) - f_{(t)}(x))\, dt \\ &= \int_0^1 \Big(\sum_{x : f(x) \geq g(x)} (f_{(t)}(x) - g_{(t)}(x)) \\ &\quad + \sum_{x : f(x) < g(x)} (g_{(t)}(x) - f_{(t)}(x)) \Big)\, dt \\ &= \int_0^1 \|f_{(t)} - g_{(t)}\|_1\, dt \geq \int_0^1 L_1(f_{(t)}, \mathcal{M})\, dt. \end{aligned}$$

The last equality above holds because for all functions f, g and $x \in D$, the inequality $f(x) \geq g(x)$ holds iff for all thresholds $t \in [0, 1]$, it holds that $f_{(t)}(x) \geq g_{(t)}(x)$. □

We use our characterization in reductions for L_1-testing (Lemma 2.2) and distance approximation. The second reduction and both proofs appear in the full version.

LEMMA 2.2. *If T is a nonadaptive 1-sided error ε-test for monotonicity of functions $f : D \to \{0,1\}$ then it is also a nonadaptive 1-sided error ε-test w.r.t. the L_1 distance for monotonicity of functions $f : D \to [0,1]$.*

2.2 Nonadaptive Tester over Hypergrids

Before proving Theorem 1.3, we state one of the building blocks—the probabilistic inequality that leads to an improvement on Levin's work investment strategy. To motivate the result, we summarize Goldreich's explanation [16] on how it can be used and instantiate it with our scenario.

Suppose an algorithm needs to find some "evidence" (e.g., a violation of monotonicity) using as little work as possible. It can select an element (e.g., a line) according to some distribution $\mathcal{D}$ and invest some work in it to discover the evidence. All elements e have different quality $q(e) \in [0,1]$, not known in advance (e.g., the distance from the line to monotonicity). To extract evidence from element e (e.g., find a violated pair on the line), the algorithm must invest work which is higher when the quality $q(e)$ is lower (e.g., work inversely proportional[7] to $q(e)$). Suppose $\mathbb{E}_{e \leftarrow \mathcal{D}}[q(e)] \geq \alpha$. What is a good work investment strategy for the algorithm?

This question can be formalized as follows. The algorithm's strategy is the list $(s_1, \ldots, s_k)$ of k positive integers, indicating that the algorithm will invest work s_i in the ith element. In the ith step, the algorithm selects an element e_i according to its distribution. If s_i is at least the work needed for item e_i, the algorithm wins. If it does not win in k steps, it loses. The cost of the strategy is $\sum_{i=1}^{k} s_i$, the total work the algorithm decided to invest. What is the minimum cost of a strategy that guarantees that the algorithm wins with constant probability (specifically, say, probability at least 2/3)? A good strategy is suggested by the following probabilistic inequality (a strengthening of [16, Fact A.1]). In this inequality, X represents the random variable equal to $q(e)$ when e is selected according to $\mathcal{D}$.

LEMMA 2.3. *Let X be a random variable that takes values in $[0,1]$. Suppose $\mathbb{E}[X] \geq \alpha$, where $\alpha \leq 1/2$, and let $t = \lceil 3 \log \frac{1}{\alpha} \rceil$. Let $\delta \in (0,1)$ be the desired probability of error. For all $j \in [t]$, let $p_j = \Pr[X \geq 2^{-j}]$ and $k_j = \frac{4 \ln 1/\delta}{2^j \alpha}$. Then*

$$\prod_{j=1}^{t} (1 - p_j)^{k_j} \leq \delta.$$

That is, for each $j \in [t]$, the algorithm can set k_j of the s_i's to be equal to the work needed for elements with quality 2^{-j} (or higher). Then the probability that the algorithm loses in all rounds is at most δ. This strategy achieves a reduction by a factor of t for the number of s_i's that need to be devoted to elements of quality 2^{-j}, compared to Levin's strategy, as explained by Goldreich. For example, when the work is inversely proportional to the quality, k_j of the s_i's would be set to 2^j. Then the total cost of the strategy for constant δ is $\sum_{j \in [t]} O(1/(2^j \alpha)) \cdot 2^j = O\left(\frac{1}{\alpha} \log \frac{1}{\alpha}\right)$. This is a reduction by a factor of $\log \frac{1}{\alpha}$ in total work, compared to Levin's strategy.

[7]The relationship between work and quality is different in our application to monotonicity, but this one is most frequently used. Also, in the application to monotonicity, there is no certainty that if you invest the specified amount of work, you will find a witness; rather, the probability of finding a witness increases with the amount of work you invest.

PROOF OF LEMMA 2.3. It is enough to prove the following inequality (whose proof appears in the full version):

$$\sum_{j=1}^{t} \frac{p_j}{2^j} \geq \frac{\alpha}{4}. \tag{1}$$

It implies the lemma since $\prod_{j=1}^{t} (1-p_j)^{k_j} \leq \prod_{j=1}^{t} e^{-p_j \cdot k_j}$ $= e^{-\sum_{j=1}^{t} p_j \cdot k_j} = e^{-\sum_{j=1}^{t} \frac{p_j}{2^j \alpha} \cdot (4 \ln 1/\delta)} \leq e^{-\frac{1}{4} \cdot (4 \ln 1/\delta)} = \delta$. The first inequality in this calculation follows from the fact that $1 - x \leq e^x$ and the last inequality follows from (1). □

PROOF OF THEOREM 1.3. By Lemma 2.1, it suffices to prove Theorem 1.3 for the special case when the input function f is Boolean. Our monotonicity tester for functions $f : [n]^d \to \{0,1\}$, like that in [9], works by picking a random axis-parallel line, querying a few random points on it and checking if they violate monotonicity. The difference is in the choice of the number of lines and points to query and in a more efficient check for monotonicity violations.

We start by establishing notation for axis-parallel lines. For $i \in [d]$, let $e^i \in [n]^d$ be 1 on the ith coordinate and 0 on the remaining coordinates. Then for every dimension $i \in [d]$ and $\alpha \in [n]^d$ with $\alpha_i = 0$, the *line along dimension i with position α* is the set $\{\alpha + x_i \cdot e^i \mid x_i \in [n]\}$. Let $\mathcal{L}_{n,d}$ be the set of all dn^{d-1} axis-parallel lines in $[n]^d$. Let $d_{\mathcal{M}}(f)$ denote the (relative) distance to monotonicity of a function f. For two functions f, g we denote the distance between f and g by $dist(f,g) = \Pr_x[f(x) \neq g(x)]$.

Algorithm 1: Nonadaptive monotonicity tester.

input : parameters n, d and ε; oracle access to $f : [n]^d \to \{0,1\}$.

for $j = 1$ **to** $\lceil 3 \log(2d/\varepsilon) \rceil$ **do**
  repeat $\lceil (8 \ln 9) d/(2^j \varepsilon) \rceil$ times:
    Sample a uniformly random line ℓ from $\mathcal{L}_{n,d}$.
    Query a set Q of $10 \cdot 2^j$ points uniformly from ℓ.
    Suppose ℓ is a line along dimension i.
    Let $x = \max_{z \in Q : f(z) = 0} z_i$; $y = \min_{z \in Q : f(z) = 1} z_i$.
    if $x > y$ **then reject**
accept

Consider the test presented in Algorithm 1. Clearly, its query complexity and running time are $O\left(\frac{d}{\varepsilon} \log \frac{d}{\varepsilon}\right)$. It is nonadaptive and always accepts all monotone functions. It remains to show that functions that are ε-far from monotone are rejected with probability at least 2/3. We use the following two lemmas from [9] in our analysis of this case.

LEMMA 2.4 ([9, LEMMA 6(2)]). *For $f : [n]^d \to \{0,1\}$, $\mathbb{E}_{\ell \leftarrow \mathcal{L}_{n,d}}[d_{\mathcal{M}}(f|_\ell)] \geq \frac{d_{\mathcal{M}}(f)}{2d}$.*

LEMMA 2.5 ([9, LEMMA 16]). *For $f : [n] \to \{0,1\}$ if $d_{\mathcal{M}}(f) = \gamma$ then for a random sample $Q \subseteq [n]$ of size k, the probability that $f|_Q$ is not monotone is at least $(1 - e^{-\gamma k/4})^2$.*

Consider the behavior of Algorithm 1 on an input function f that is ε-far from monotone. We apply our improvement

to Levin's strategy (Lemma 2.3) with the random variable $X = d_{\mathcal{M}}(f|_\ell)$, where line ℓ is selected uniformly from $\mathcal{L}_{n,d}$, and with probability of error $\delta = 1/9$. By Lemma 2.4, the expectation $\mathbb{E}[X] \geq \frac{\varepsilon}{2d}$. Thus, in Lemma 2.3, parameter α is set to $\frac{\varepsilon}{2d}$, parameter t is set to $\lceil 3 \log \frac{2d}{\varepsilon} \rceil$, and for all $j \in [t]$, the parameter $k_j = \frac{(8 \ln 9) \cdot d}{2^j \varepsilon}$. By Lemma 2.3, with probability at least 8/9, in at least one iteration of Step 3, Algorithm 1 will sample a line ℓ satisfying $d_{\mathcal{M}}(f|_\ell) \geq 2^{-j}$. Conditioned on sampling such a line, by Lemma 2.5, the sample Q taken in Step 4 of Algorithm 1 results in a non-monotone restriction $f|_Q$ with probability at least 3/4. Finally, Step 6 of Algorithm 1 rejects iff $f|_Q$ is not monotone. Thus, for the index j given by Lemma 2.3, the probability that Algorithm 1 rejects f in the jth iteration of the **for** loop is at least $\frac{8}{9} \cdot \frac{3}{4} = \frac{2}{3}$, as required. □

2.3 Adaptive Tester over Hypergrids

THEOREM 2.6. *The query complexity of ε-testing monotonicity of functions $f\colon [n]^d \to \{0,1\}$ (with one-sided error) is $O\big(d2^d \log^{d-1} \frac{1}{\varepsilon} + \frac{d^2 \log d}{\varepsilon}\big)$. Specifically, for constant d, the query complexity is $O\left(\frac{1}{\varepsilon}\right)$.*

Recall from Section 1.2.1 that our adaptive tester is based on an algorithm that learns the class of monotone Boolean functions over $[n]^d$ in the sense of Definition 2.3 below. The learner is presented in Section 2.3.1. Transformation from the learner to the tester is given in Lemma 2.11. This transformation, together with our learner, immediately implies Theorem 2.6 for the special case of $d = 2$. The general test is presented in Section 2.3.3.

2.3.1 Partial learner of monotone Boolean functions

DEFINITION 2.2. *An ε-*partial function *g with domain D and range R is a function $g : D \to R \cup \{?\}$ that satisfies $\Pr_{x \in D}[g(x) = ?] \leq \varepsilon$. An ε-partial function g* agrees *with function f if $g(x) = f(x)$ for all x on which $g(x) \neq ?$. Given a function class $\mathcal{C}$, let $\mathcal{C}_\varepsilon$ denote the class of ε-partial functions, each of which agrees with some function in $\mathcal{C}$.*

DEFINITION 2.3. *An ε-*partial learner *for a function class $\mathcal{C}$ is an algorithm that, given a parameter ε and oracle access to a function f, outputs a hypothesis $g \in \mathcal{C}_\varepsilon$ or fails. Moreover, if $f \in \mathcal{C}$ then it outputs g that agrees with f.*

THEOREM 2.7. *Algorithm 2 is a $d/2^t$-partial learner for the class of monotone Boolean functions over $[n]^d$. It makes $O(d2^{t(d-1)+d})$ queries.*

PROOF. Our learner (Algorithm 2) constructs a d-dimensional quadtree representation of (a partial function that agrees with) the input function f. Each node in the tree is marked 0, 1 or ?. The learner stops after constructing t tree levels. The quadtree is a representation of a partial Boolean function g on $[n]^d$ in the following sense. Each point x in the domain is contained in exactly one leaf, and $g(x)$ is equal to the value the leaf is marked with.

LEMMA 2.8. *In the quadtree returned by Algorithm 2, for all j, at most $d2^{j(d-1)}$ nodes in level j are marked ?.*

PROOF. Fix j. Consider the set $S = \{\frac{v[\text{hi}]}{n/2^j} \mid v \text{ is a node of level } j \text{ marked with } ?\}$.

Algorithm 2: Partial learner of monotone functions

input : parameters n, d and ε; depth t; oracle access to a monotone $f\colon [n]^d \to \{0,1\}$.
output: a d-dimensional quadtree representation of f.

1 Create quadtree root marked with ? and holding $[n]^d$, i.e., set root[low]$= \mathbf{1}_d$ and root[hi]$= n \cdot \mathbf{1}_d$. `// 1_d denotes the d-dimensional vector of 1s.`
2 **for** $j = 1$ **to** t **do**
  `// Create nodes of level j in the quadtree.`
3 Set $len = \frac{n}{2^j}$.
4 **foreach** leaf v of the quadtree marked with ? **do**
    `// Create 2^d children of v.`
5 **foreach** vector $\mathbf{k} \in \{0,1\}^d$ **do**
6 Create child $v_{\mathbf{k}}$ of v marked with ?.
7 Set $v_{\mathbf{k}}[\text{low}] = v[\text{low}] + len \cdot \mathbf{k}$ and $v_{\mathbf{k}}[\text{hi}] = v_{\mathbf{k}}[\text{low}] + (len - 1)\mathbf{1}_d$.
8 **if** $f(v_k[\text{hi}]) = 0$ **then** mark child v_k with 0.
9 **if** $f(v_k[\text{low}]) = 1$ **then** mark child v_k with 1.
10 **return** the quadtree

DEFINITION 2.4. *Let $x, y \in [n]^d$. We say that y* fully dominates *x if $x_i < y_i$ for all $i \in [d]$.*

Consider two nodes u and v at level j of the quadtree, such that v[hi] fully dominates u[hi]. Observe that at most one of u and v can be marked with ?. Indeed, if $f(u[\text{hi}]) = 0$ then u is marked with 0. Otherwise, $f(u[\text{hi}]) = 1$. By monotonicity of f, since $u[\text{hi}] \prec v[\text{low}]$, it follows that $f(v[\text{low}]) = 1$. That is, v is marked with 1. Thus, for no elements $x, y \in S$, element y fully dominates element x.

Finally, observe that $S \subseteq [m]^d$, where $m = 2^j$. The lemma follows from Claim 2.9 that bounds the number of elements in a set that satisfies the observed conditions. □

CLAIM 2.9. *Let $S \subseteq [m]^d$, such that for no elements $x, y \in S$, element y fully dominates element x. Then $|S| \leq dm^{d-1}$.*

PROOF. Let $A = \{a \in [m]^d \mid a_i = 1 \text{ for some } i \in [d]\}$. We can partition $[m]^d$ into chains $C_a = \{(a + c \cdot \mathbf{1}_d) \in [m]^d \mid c \in \mathbb{N}\}$, where $a \in A$. Note that for all $a \in A$ and all distinct $x, y \in C_a$, element y fully dominates element x. Thus, S can contain at most 1 element in each C_a. Consequently,

$$|S| \leq |A| \leq d \cdot m^{d-1}.$$

The last inequality holds because there are d ways to choose i such that $a_i = 1$ and at most m^{d-1} ways to set the remaining coordinates. □

Correctness of Algorithm 2. Each node at level t holds $1/2^{dt}$ fraction of the domain points. Therefore, by Lemma 2.8, the fraction of the domain points that are associated with ? leaves of the quadtree is at most $\frac{d2^{t(d-1)}}{2^{td}} = \frac{d}{2^t}$.

If function f is monotone, the remaining points are learned correctly because for each domain point x associated with a node that is marked 0 or 1, we have a witness that implies the corresponding value for $f(x)$ by monotonicity of f.

Complexity of Algorithm 2. The learner makes 2^{d+1} queries for each node marked with ?: two per child. By Lemma 2.8, there at most $\sum_{j=0}^{t} d2^{j(d-1)} \leq 2d \cdot 2^{t(d-1)}$ nodes marked ?, so it makes $O(d2^{t(d-1)+d})$ queries. This completes the proof of Theorem 2.7. □

COROLLARY 2.10. *There is an ε-partial learner for the class of monotone Boolean functions over $[n]^2$ that makes $O\left(\frac{1}{\varepsilon}\right)$ queries. Over $[n]^d$, there is a $\frac{d}{\log\frac{1}{\varepsilon}}$-partial learner that makes $O\big(d2^d\log^{d-1}\frac{1}{\varepsilon}\big)$ queries.*

PROOF. To get the first statement, we run Algorithm 2 with $d = 2$ and depth $t = \lceil\log\frac{1}{\varepsilon}\rceil + 1$. For the second statement, we set $t = \lceil\log\log\frac{1}{\varepsilon}\rceil$. □

2.3.2 Optimal tester for functions $[n]^2 \to \{0,1\}$

For the special case of $d = 2$, we can easily get the tester claimed in Theorem 2.6 from the learner of Theorem 2.7 by applying the following reduction from testing to learning.

LEMMA 2.11. *If there is an ε-partial learner for a function class $\mathcal{C}$ that makes $q(\varepsilon)$ queries then $\mathcal{C}$ can be ε-tested with 1-sided error with $q(\varepsilon/2) + O\left(\frac{1}{\varepsilon}\right)$ queries.*

PROOF. To test if a function $f \in \mathcal{C}$, we run the learner with parameter $\varepsilon/2$. If it fails to output a hypothesis function g, we reject. Otherwise, we select $\lceil\frac{2\ln 3}{\varepsilon}\rceil$ points $x \in D$ uniformly at random, and query both f and g on the sample. If we find a point x on which $g(x) \neq ?$, but $f(x) \neq g(x)$, we reject. Otherwise, we accept.

If a function $f \in \mathcal{C}$, it is accepted because the learner will output a hypothesis g which agrees with f. If f is ε-far from $\mathcal{C}$ then any $g \in \mathcal{C}_{\varepsilon/2}$ will differ from f on at least ε domain points, at most $\varepsilon/2$ of which can be mapped to ? by g. Thus, for at least an $\varepsilon/2$ fraction of the domain points $g(x) \neq ?$ and $g(x) \neq f(x)$. Such a point will be selected with probability at least 2/3. □

2.3.3 Optimal tester for functions $f\colon [n]^d \to \{0,1\}$

PROOF OF THEOREM 2.6. Our tester is Algorithm 3.

Algorithm 3: Adaptive monotonicity tester.

input : parameters $d \geq 3$, n and ε; oracle access to $f\colon [n]^d \to \{0,1\}$.

1 Let g be the $\frac{d}{\log\frac{1}{\varepsilon}}$-partial function returned by the learner from Corollary 2.10 run on f.
2 **repeat** $\lceil\frac{2\log 3}{\varepsilon}\rceil$ times:
3 Sample a uniform $x \in [n]^d$, query $f(x)$ and **reject** if $g(x) \neq ?$ and $g(x) \neq f(x)$.
4 Run Algorithm 1 with parameters $n, d, \varepsilon/2$ as follows:
5 **foreach** *point x queried by the algorithm* **do**
6 Query $f(x)$ only if $g(x) = ?$; otherwise, substitute $g(x)$ for $f(x)$.
7 **accept** if Algorithm 1 accepts, **reject** if it rejects.

Correctness. Algorithm 3 always accepts a monotone function because the learner produces a correct partial function g, there are no inconsistencies between g and f, and Algorithm 1 always accepts monotone functions. Now suppose f is ε-far from monotone. Let $h : [n]^d \to \{0,1\}$ be the function defined as follows: $h(x) = f(x)$ if $g(x) = ?$, and $h(x) = g(x)$ otherwise. If $dist(f,h) > \varepsilon/2$ then Step 2 of Algorithm 3 rejects with probability at least 2/3. Otherwise, h is $\varepsilon/2$-far from monotone. Consequently, it is rejected by Step 4 with probability at least 2/3.

Query complexity. By Corollary 2.10, the learning step uses $O\big(d2^d\log^{d-1}\frac{1}{\varepsilon}\big)$ queries. Step 2 (that checks whether g agrees with f) uses $O(1/\varepsilon)$ queries. It remains to analyze Step 4. Algorithm 1 queries $O(\frac{d}{\varepsilon}\log\frac{d}{\varepsilon})$ points. Out of these, only the points mapped to ? by g are queried in Step 4. Since g is a $d/\log\frac{1}{\varepsilon}$-partial function, it maps at most $d/\log\frac{1}{\varepsilon}$ fraction of its domain to ?. The expected number of queries made by Step 4 is $O(\frac{d}{\varepsilon}\log\frac{d}{\varepsilon})$ times the probability that the ith queried point x is mapped to ? by g. This probability is at most $d/\log\frac{1}{\varepsilon}$, as all points in $[n]^d$ are equally likely to be the ith query (for all i). So, the expected query complexity is $O(\frac{d}{\varepsilon}\log\frac{d}{\varepsilon})\cdot\frac{d}{\log\frac{1}{\varepsilon}} = O(\frac{d^2}{\varepsilon} + \frac{d^2\log d}{\varepsilon\log 1/\varepsilon}) = O(\frac{d^2\log d}{\varepsilon})$. □

2.4 A Lower Bound for Functions on Grid

THEOREM 2.12. *Every 1-sided error nonadaptive ε-test for monotonicity of functions $f : [n]^2 \to \{0,1\}$ must make $\Omega\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ queries.*

PROOF. By standard arguments, it suffices to specify a distribution on functions which are ε-far from monotone, such that every deterministic algorithm must make $\Omega\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ queries to find a violated edge with probability at least 2/3 on inputs from this distribution. Our distribution is uniform over a *hard* set of functions, each of which corresponds to a hexagon. All pairs violated by a function in the set are contained in the corresponding hexagon. In this proof, we identify each point $(i,j) \in [n]^2$ in the domain of the input function with a point (x,y) with coordinates $x = i/n$ and $y = j/n$ in the Euclidian plane. We assume that n is large enough, so that the area of hexagons in $[0,1]^2$ we consider is a good approximation of the fraction of points of the form $(i/n, j/n)$ they contain.

DEFINITION 2.5. Hexagon $H^{\mathbf{x},\mathbf{y}}_{t,h}$ *with the upper left corner $(\mathbf{x},\mathbf{y})$, thickness t, and height h consists of points (x,y) :*

$$\begin{array}{rcccl} \mathbf{x}+\mathbf{y}-t & < & x+y & < & \mathbf{x}+\mathbf{y}+t, \\ \mathbf{x} & < & x & < & \mathbf{x}+h, \\ \mathbf{y}-h & < & y & < & \mathbf{y}. \end{array}$$

Now define the hexagon function $f^{\mathbf{x},\mathbf{y}}_{t,h} : [0,1]^2 \to \{0,1\}$:

$$f^{\mathbf{x},\mathbf{y}}_{t,h}(x,y) = \begin{cases} 1 - d_{\mathbf{x}+\mathbf{y}}(x,y) & \text{if } (x,y) \in H^{\mathbf{x},\mathbf{y}}_{t,h}, \\ d_{\mathbf{x}+\mathbf{y}}(x,y) & \text{otherwise,} \end{cases}$$

$$\text{where } d_a(x,y) = \begin{cases} 1 & \text{if } x+y \geq a, \\ 0 & \text{otherwise.} \end{cases}$$

Next, we specify the hexagon functions included in the hard set. The hard set is a union of *levels*, and each level is a union of *diagonals*. Functions of the same level are defined by equal non-overlapping hexagons. Levels are indexed by integers starting from 0. Let $t_0 = \sqrt{\varepsilon}/\log\frac{1}{\varepsilon}$. In level i, hexagons have thickness $t_i = t_0\cdot 2^{-i}$ and height $h_i = 2\varepsilon/t_i$. We include levels $i \geq 0$ with thickness $t_i \geq 4\varepsilon$.

Each level i is further subdivided into *diagonals* indexed by an integer $j \in (1, 1/t_i)$. In diagonal j, the coordinates of the upper left corners of the hexagons satisfy $\mathbf{x}+\mathbf{y} = (2j+1)t_i$. It remains to specify the functions that are contained in each diagonal. Intuitively, we pack the maximum possible number of hexagons into each diagonal, while leaving sufficient separation between them. If $\mathbf{x}+\mathbf{y} = (2j+1)t_i \leq 1$, we restrict $\mathbf{x}$ to integer multiples of $h_i + \sqrt{\varepsilon}$, and for $\mathbf{x}+\mathbf{y} = (2j+1)t_i > 1$, we restrict $1-\mathbf{y}$ to integer multiples of $h_i + \sqrt{\varepsilon}$. In both cases, the projections of the

hexagons of these functions onto an axis form disjoint intervals of length h_i that are separated by gaps of length $\sqrt{\varepsilon}$. Finally, only if a hexagon $H^{\mathbf{x},\mathbf{y}}_{t,h}$ is fully contained in $[0,1]^2$, the corresponding function $f^{\mathbf{x},\mathbf{y}}_{t,h}$ is included in the hard set.

This construction is analyzed in the full version. □

3. L_1-TESTING OF LIPSCHITZ PROPERTY

3.1 Distance to Lipschitz: Characterization

In this section, we recall some basic definitions from [21, 9, 5] and present our characterization of the L_1 distance to the Lipschitz property.

DEFINITION 3.1. *Let $f : D \to \mathbb{R}$ be a function with a finite domain D, equipped with distance d_D, and let x, y be points in D. The pair (x, y) is* violated *by f if $|f(x)-f(y)| > d_D(x, y)$. The* violation score *of (x, y), denoted $\mathrm{vs}_f(x, y)$, is $|f(x)-f(y)|-d_D(x, y)$ if it is violated and 0 otherwise. The* violation graph *G_f of function f is the weighted undirected graph with vertex set D, which contains an edge (x, y) of weight $\mathrm{vs}_f(x, y)$ for each violated pair of points $x, y \in D$.*

Let $\mathcal{L}ip$ be the set of Lipschitz functions $f : D \to \mathbb{R}$. The following lemma characterizes $L_1(f, \mathcal{L}ip)$, the absolute L_1 distance from f to $\mathcal{L}ip$, in terms of matchings in G_f.

LEMMA 3.1 (CHARACTERIZATION). *Consider a function $f : D \to \mathbb{R}$, where D is a finite metric space. Let M be a maximum weight matching in G_f. Then $L_1(f, \mathcal{L}ip) = \mathrm{vs}_f(M)$, where $\mathrm{vs}_f(M)$ denotes the weight of M.*

PROOF. It has already been observed that $L_1(f, \mathcal{L}ip) \geq \mathrm{vs}_f(M)$ for any matching M in G_f [2, Lemma 2.3]. To show the second needed inequality, let g be a Lipschitz function closest to f, that is, satisfying $L_1(f, g) = L_1(f, \mathcal{L}ip)$. We will construct a matching M in G_f such that $L_1(f, g) = \mathrm{vs}_f(M)$. We start by constructing a matching M' of weight $L_1(f, g)$ in a bipartite graph related to G_f, and later transform it to the matching of the same weight in G_f.

DEFINITION 3.2. *For each operation $op \in \{<, >, =\}$, define a point set $V_{op} = \{x \in D \mid f(x)\ op\ g(x)\}$.*

DEFINITION 3.3 (BIPARTITE GRAPH BG_f). *BG_f is a bipartite weighted graph. The nodes of BG_f are points in D, except that we make two copies, called $x_\leq$ and $x_\geq$, of every point $x \in V_=$. Nodes are partitioned into $V_\geq$ and $V_\leq$, where part $V_\geq = V_> \cup \{x_\geq \mid x \in V_=\}$ and, similarly, part $V_\leq = V_< \cup \{x_\leq \mid x \in V_=\}$. The set BE_f of edges of BG_f consists of pairs $(x, y) \in V_> \times V_\leq \cup V_\geq \times V_<$, such that*

$$g(x) - g(y) = d_D(x, y).$$

Metric d_D is extended to duplicates: $d_D(x_\leq, y) = d_D(x_\geq, y) = d_D(x, y)$, and the weights $\mathrm{vs}_f(x, y)$ are defined as before.

Observe that for every edge (x, y) in BG_f, by definition,

$$f(x) \geq g(x) > g(y) \geq f(y) \text{ and} \tag{2}$$

$$\begin{aligned} \mathrm{vs}_f(x, y) &= f(x) - f(y) - d_D(x, y) \\ &= f(x) - f(y) - (g(x) - g(y)) \\ &= |f(x) - g(x)| + |f(y) - g(y)|. \end{aligned} \tag{3}$$

Later, we will show that BG_f contains a matching M' that matches every $x \in V_< \cup V_>$. By (3),

$$\mathrm{vs}_f(M') = \sum_{(x,y)\in M'} \mathrm{vs}_f(x, y) = \sum_{x \text{ is matched in } M'} |f(x) - g(x)|$$

$$= \sum_{x \in V_< \cup V_>} |f(x) - g(x)| = \|f - g\|_1 = L_1(f, \mathcal{L}ip).$$

Next we show how to transform a matching M' in BG_f into a matching M in G_f, such that $\mathrm{vs}_f(M) = \mathrm{vs}_f(M')$. We first remove from M' all edges of weight 0. Then we replace each $x_\leq$ and $x_\geq$ with x. Finally, we handle the cases when both $x_\leq$ and $x_\geq$ were replaced with x. If this happens, M' contains matches $(y, x_\leq)$ and $(x_\geq, z)$, and we replace them with a single match (y, z). By (2), $f(y) > f(x) > f(z)$. Since (y, x) and (x, z) are violated pairs, $\mathrm{vs}_f(y, z) = \mathrm{vs}_f(y, x) + \mathrm{vs}_f(x, z)$. Thus, $\mathrm{vs}_f(M) = \mathrm{vs}_f(M')$, as claimed.

Now it suffices to show that BG_f contains a matching M' that matches every $x \in V_< \cup V_>$. By Hall's marriage theorem, it enough to show that whenever $A \subseteq V_<$ or $A \subseteq V_>$ we have $|A| \leq |N(A)|$, where $N(A)$ is the set of all neighbors of A in BG_f. Suppose to the contrary that the marriage condition does not hold for some set, and w.l.o.g. suppose it is a subset of $V_>$. Let $A \subset V_>$ be the largest set satisfying $|A| > |N(A)|$.

CLAIM 3.2. *If $x, y \in V_>$ and $g(x) - g(y) = d_D(x, y)$ then $N(y) \subseteq N(x)$.*

PROOF. Suppose that $z \in N(y)$, i.e., $f(z) \leq g(z)$ and $g(y) - g(z) = d_D(y, z)$. Using the triangle inequality, we get

$$\begin{aligned} g(x) - g(z) &= [g(x) - g(y)] + [g(y) - g(z)] \\ &= d_D(x, y) + d_D(y, z) \geq d_D(x, z). \end{aligned}$$

Since g is Lipschitz, $g(x)-g(z) \leq d_D(x, z)$. Therefore, $g(x)-g(z) = d_D(x, z)$, and (x, z) is an edge of BG_f. □

Since A is the largest set that fails the marriage condition, if $x \in A$, $y \in V_>$ and $g(x) - g(y) = d_D(x, y)$ then $y \in A$. Similarly, we can argue that if $x \in N(A)$, $y \in V_\leq$ and $g(x) - g(y) = d_D(x, y)$ then $y \in N(A)$.

Thus, $g(x) - g(y) < d_D(x, y)$ for all $x \in A \cup N(A)$ and $y \notin A \cup N(A)$. Consequently, for some $\delta > 0$, if we increase $g(x)$ by δ for every $x \in A \cup N(A)$ then g remains Lipschitz. This decreases $L_1(f, g)$ by $\delta(|A|-|N(A)|$, a contradiction to g being the closest Lipschitz function to f in L_1 distance. □

3.2 c-Lipschitz Tester for Hypergrids

In this section, we present our c-Lipschitz test for functions on hypergrid domains and prove Theorem 1.4. Observe that testing the c-Lipschitz property of functions with range $[0, 1]$ is equivalent to testing the Lipschitz property of functions with range $[0, 1/c]$ over the same domain. To see this, first note that function f is c-Lipschitz iff function f/c is Lipschitz. Second, function f is ε-far from being c-Lipschitz iff f/c is ε-far from being Lipschitz, provided that the relative L_1 distance between functions scaled correctly: namely, for functions $f, g : D \to [0, r]$, we define $d_1(f, g) = \frac{\|f-g\|_1}{|D|\cdot r}$. Thus, we can restate Theorem 1.4 as follows.

THEOREM 3.3 (THM. 1.4 RESTATED). *Let $n, d \in \mathbb{N}$, $\varepsilon \in (0, 1)$, $r \in [1, \infty)$. The time complexity of L_1-testing the Lipschitz property of functions $f : [n]^d \to [0, r]$ (nonadaptively and with 1-sided error) with proximity parameter ε is $O\left(\frac{d}{\varepsilon}\right)$.*

Next we present the test (Algorithm 4) with the complexity claimed in Theorem 3.3. We use the notation for axis-parallel lines established before Algorithm 1.

Algorithm 4: Nonadaptive Lipschitz tester.

input : parameters n, d and ε; oracle access to $f\colon [n]^d \to [0, r]$.

repeat $\lceil \frac{d \cdot 8 \ln 3}{\varepsilon} \rceil$ times:
  Sample a uniform line ℓ from $\mathcal{L}_{n,d}$. // $\mathcal{L}_{n,d}$ is the set of axis-parallel lines in $[n]^d$.
  Let P_ℓ^r be the set of (unordered) pairs (x, y) of points from ℓ such that $\|x - y\|_1 \leq r$.
  Query a uniformly random pair of points from P_ℓ^{r-1}.
  if $|f(x) - f(y)| > \|x - y\|_1$ **then reject**
accept

Algorithm 4 is nonadaptive. It always accepts all Lipschitz functions. It remains to prove that a function that is ε-far from Lipschitz in L_1 distance is rejected with probability at least 2/3. Since the algorithm is simply picking pairs (x, y) from a certain set and checking if the Lipschitz property is violated on the selected pairs, it suffices to show that a large fraction of the considered pairs (x, y) is violated by any function that is ε-far from Lipschitz. Observe that for each $\ell \in \mathcal{L}_{n,d}$, the number of pairs x, y on the line ℓ is $n(n-1)/2$. If $r < n$ then $|P_\ell^{r-1}| \leq n(r-1)$. I.e., $|P_\ell^{r-1}| \leq n \cdot \min\{n-1, r-1\}$. Note that $|f(x) - f(y)| > \|x - y\|_1$ implies that $\|x - y\|_1 \leq r - 1$. I.e., all violated pairs on the line ℓ are in P_ℓ^{r-1}. To complete the analysis of Algorithm 4, it suffices to prove the following lemma. (See full version.)

LEMMA 3.4. *Let $\mathcal{P}_{n,d}$ bet the set of pairs $x, y \in [n]^d$, where x and y differ in exactly one coordinate. Consider a function $f : [n]^d \to [0, r]$ that is ε-far from Lipschitz w.r.t. the L_1 distance. Then the number of pairs in $\mathcal{P}_{n,d}$ violated by f is at least $\frac{\varepsilon n}{8d} \cdot \min\{n-1, r-1\} \cdot |\mathcal{L}_{n,d}|$.*

Acknowledgements

We would like to thank Kenneth Clarkson, Jan Vondrak and Vitaly Feldman for helpful discussions and Adam Smith for comments on this document.

References

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. *Random Struct. Algorithms*, 31(3):371–383, 2007.

[2] P. Awasthi, M. Jha, M. Molinaro, and S. Raskhodnikova. Testing Lipschitz functions on hypergrid domains. In *APPROX-RANDOM*, pages 387–398, 2012.

[3] T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. Testing closeness of discrete distributions. *J. ACM*, 60(1):4, 2013.

[4] E. Blais, S. Raskhodnikova, and G. Yaroslavtsev. Lower bounds for testing properties of functions over hypergrid domains. In *CCC*, 2014.

[5] D. Chakrabarty and C. Seshadhri. Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *STOC*, pages 419–428, 2013.

[6] D. Chakrabarty and C. Seshadhri. An optimal lower bound for monotonicity testing over hypergrids. In *APPROX-RANDOM*, pages 425–435, 2013.

[7] D. Chakrabarty, K. Dixit, M. Jha, and C. Seshadhri. Optimal lower bounds for Lipschitz testing via monotonicity. *Private communication*, 2013.

[8] K. Dixit, M. Jha, S. Raskhodnikova, and A. Thakurta. Testing the Lipschitz property over product distributions with applications to data privacy. In *TCC*, pages 418–436, 2013.

[9] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. In *RANDOM*, pages 97–108, 1999.

[10] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[11] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *J. Comput. Syst. Sci.*, 60 (3):717–751, 2000.

[12] S. Fattal and D. Ron. Approximating the distance to convexity. *Available at:* `http://www.eng.tau.ac.il/~danar/Public-pdf/app-conv.pdf`, 2007.

[13] S. Fattal and D. Ron. Approximating the distance to monotonicity in high dimensions. *ACM Transactions on Algorithms*, 6(3), 2010.

[14] V. Feldman and J. Vondrák. Optimal bounds on approximation of submodular and XOS functions by juntas. In *FOCS*, pages 227–236, 2013.

[15] E. Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004.

[16] O. Goldreich. On multiple input problems in property testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:67, 2013.

[17] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

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

[19] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

[20] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Inf. Comput.*, 100(1):78–150, 1992.

[21] M. Jha and S. Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. *SIAM J. Comput.*, 42(2):700–731, 2013.

[22] L. A. Levin. One-way functions and pseudorandom generators. In *STOC*, pages 363–365, 1985.

[23] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *J. Comput. Syst. Sci.*, 72(6):1012–1042, 2006.

[24] L. Rademacher and S. Vempala. Testing geometric convexity. In *FSTTCS*, pages 469–480, 2004.

[25] S. Raskhodnikova. Approximate testing of visual properties. In *RANDOM-APPROX*, pages 370–381, 2003.

[26] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[27] M. Saks and C. Seshadhri. Estimating the longest increasing sequence in polylogarithmic time. In *FOCS*, pages 458–467, 2010.