Testing and Reconstruction of Lipschitz Functions with Applications to Data Privacy[*][†]

Madhav Jha
Pennsylvania State University
mxj201@cse.psu.edu

Sofya Raskhodnikova
Pennsylvania State University
sofya@cse.psu.edu

Abstract— A function $f : D \to R$ has *Lipschitz* constant c if $d_R(f(x), f(y)) \leq c \cdot d_D(x, y)$ for all x, y in D, where d_R and d_D denote the distance metrics on the range and domain of f, respectively. We say a function is *Lipschitz* if it has Lipschitz constant 1. (Note that rescaling by a factor of $1/c$ converts a function with a Lipschitz constant c into a Lipschitz function.) Intuitively, a Lipschitz constant of f is a bound on how sensitive f is to small changes in its input.

We initiate the study of testing and local reconstruction of the Lipschitz property of functions. A *property tester*, given a parameter ϵ, has to distinguish functions with the property (in this case, Lipschitz) from functions that are ϵ-far from having the property, that is, differ from every function with the property on at least an ϵ fraction of the domain. A *local filter* reconstructs a desired property (in this case, Lipschitz) in the following sense: given an arbitrary function f and a query x, it returns $g(x)$, where the resulting function g satisfies the property, changing f only when necessary. If f has the property, g must be equal to f.

We consider functions over domains of the form $\{1, \dots, n\}^d$, equipped with ℓ_1 distance. We design efficient testers of the Lipschitz property for functions of the form $f : \{1, 2\}^d \to \delta\mathbb{Z}$, where $\delta \in (0, 1]$ and $\delta\mathbb{Z}$ is the set of integer multiples of δ, and of the form $f : \{1, \dots, n\} \to R$, where R is (discretely) metrically convex. We also present an efficient local filter of the Lipschitz property for functions of the form $f : \{1, \dots, n\}^d \to \mathbb{R}$. We give corresponding lower bounds on the complexity of testing and local reconstruction.

The algorithms we design have applications to program analysis and data privacy. The application to privacy is based on the fact that a function f of entries in a database of sensitive information can be released with noise of magnitude proportional to a Lipschitz constant of f, while preserving the privacy of individuals whose data is stored in the database (Dwork, McSherry, Nissim and S mith, TCC 2006). We give a differentially private mechanism, based on local filters, for releasing a function f when a purported Lipschitz constant of f is provided by a distrusted client. We show that when no reliable Lipschitz constant of f is given, previously known differentially private mechanisms have either a substantially higher running time or a higher expected error, for a large class of symmetric functions f.

Keywords-Property Testing; Lipschitz Constant; Data Privacy; Reconstruction;

[*]This work was supported by the National Science Foundation (NSF/CCF CAREER award 0845701).

[†]All missing proofs appear in the full version of this paper [14].

1. Introduction

Consider a function $f : D \to R$ mapping a metric space (D, d_D) to a metric space (R, d_R), where d_D and d_R denote the distance functions on the domain D and range R, respectively. Function f has *Lipschitz constant* c if $d_R(f(x), f(y)) \leq c \cdot d_D(x, y)$ for all x, y in D. We call such a function *c-Lipschitz* and say a function is *Lipschitz* if it is 1-Lipschitz. (Note that rescaling by a factor of $\frac{1}{c}$ converts a c-Lipschitz function into a Lipschitz function.) Intuitively, a Lipschitz constant of f is a bound on how sensitive f is to small changes in its input.

Lipschitz continuity[1] is a fundamental notion in mathematical analysis, the theory of differential equations and other areas of mathematics and computer science. A Lipschitz constant c of a given function f is used, for example, in probability theory in order to obtain tail bounds via McDiarmid's inequality [16]; in program analysis, it is considered as a measure of robustness to noise [7]; in data privacy, it is used to scale noise added to output $f(x)$ to preserve differential privacy of a database x [11]. In these three examples, one often needs to compute a Lipschitz constant of a given function f or, at least, verify that f is c-Lipschitz for a given number c. However, in general, computing a Lipschitz constant is computationally infeasible. The decision version is undecidable when f is specified by a Turing machine that computes it, and NP-hard if f is specified by a circuit. In this work, we focus on Lipschitz continuity of functions over finite domains, for which the NP-hardness statement still holds.

We initiate the study of *testing* if a function (over a finite domain) is Lipschitz, which is a relaxation of the decision problem described above. A property *tester* [19], [13] is given oracle access to an object

[1]A function is called *Lipschitz continuous* if there is a constant c for which it is c-Lipschitz.

(in this case, a function f) and a proximity parameter ϵ. It has to distinguish functions with the property (in this case, Lipschitz) from functions that are ϵ-far from having the property, that is, differ from every function with the property on at least an ϵ fraction of the domain. Intuitively, a tester for the Lipschitz property of functions provides an approximate answer to the decision problem of determining if a function is Lipschitz and is useful in some situations when obtaining an exact answer is computationally infeasible.

We also study *local reconstruction* of the Lipschitz property of functions over finite domains. This is useful in applications (in particular, to data privacy) where merely testing is not sufficient, and one needs to be able to enforce the Lipschitz property. Property-preserving data reconstruction [1] is beneficial when an algorithm, call it A, is computing on a large dataset and the algorithm's correctness is contingent upon the dataset satisfying a certain structural property. For example, A may require that its input array be sorted or, in our case, its input function be Lipschitz. In such situations, A could access its input via a *filter* that ensures that data seen by A always satisfy the desired property, modifying it at few places on the fly, if required. Suppose that A's input is represented by a function f. Then whenever A wants to access $f(x)$, it makes query x to the filter. The filter looks up the value of f on a small number of points and returns $g(x)$, where g satisfies the desired property (in our case, is Lipschitz). See Figure 1. Thus, A is computing with reconstructed data g instead of its original input f.

Local reconstruction [20] imposes an additional requirement to allow for parallel or distributed implementation of filters: the output function g must be independent of the order of the queries x to the filter. The version of local reconstruction we consider (see Definition 2.1), defined in [3], further requires that if the original input has the property, it should not be modified by the filter, i.e., if f has the property, g must be equal to f. Our application to data privacy has an unusual feature, not encountered in previous applications of filters: algorithm A needs to access its input only at one point x (corresponding to the database its holding). Nevertheless, we require *local filters*, not because of the distributed aspect they were initially developed for, but because when g depends on x, it might leak information about x and violate privacy.

Previous work on property testing and reconstruction: Property testing [13], [19] is a well-studied notion of approximation for decision problems. Properties of a wide variety of structures, including graphs, error-correcting codes, geometric sets, probability distributions, images and Boolean functions, have been investigated in this context, most of which are not directly related to the problems we consider here. A notable exception is work on testing monotonicity of functions, first considered in [12] (see, e.g., [4] and references therein), which has provided several techniques that are surprisingly useful for testing the Lipschitz property. We give relevant pointers for each of our results in Section 1.1. A unified discussion can be found in the full version of this paper.

Property preserving reconstruction [1] has been studied for monotonicity of functions [1], [20], [3], convexity of points [8], graph expansion [15] and error-correcting codes [6]. The local model is addressed in [20], [6], [3], with only [20] providing local filters, and the other two papers focusing on lower bounds. Results on filters for properties other than monotonicity of functions do not seem directly relevant to our work.

1.1. Our Results and Techniques

We study testing and local reconstruction of Lipschitz functions over discrete metric spaces. Standard notions from property testing and reconstruction are introduced in Section 2. Throughout the paper, we use $[n]$ to denote $\{1, \ldots, n\}$. We represent each domain by a graph G equipped with the shortest path distance d_G. Specifically, we consider functions over domains $\{0,1\}^d$, $[n]$ and $[n]^d$, equipped with ℓ_1 distance. We refer to the domains of our functions by specifying the underlying graph that captures the distances between points in the domain. Specifically, $\{0,1\}^d$ is referred to as the hypercube $\mathcal{H}_d$, $[n]$ as the line $\mathcal{L}_n$ and $[n]^d$ as the hypergrid $\mathcal{H}_{n,d}$. The hypergrid $\mathcal{H}_{n,d}$ has vertex set $[n]^d$ and edge set $\{\{x,y\} : \exists$ unique $i \in [d]$ such that $|y_i - x_i| = 1$ and for $j \neq i, y_j = x_j\}$. The line and the hypercube are the special cases of the hypergrid for $d = 1$ and $n = 2$, respectively, with vertices of the hypercube renumbered as $\{0,1\}^d$ instead of $\{1,2\}^d$.

Testing the Lipschitz property on the hypercube: We design efficient testers of the Lipschitz property for functions over the hypercube $\mathcal{H}_d$ and the line $\mathcal{L}_n$ and prove corresponding lower bounds.

The following theorem, proved in Section 3, gives our main technical result: a tester for the Lipschitz property of functions of the form $f : \mathcal{H}_d \to \delta\mathbb{Z}$, where $\delta \in (0,1]$ and $\delta\mathbb{Z}$ is the set of integer multiples of δ. Its performance is better when a small upper bound on the image diameter of the input function is known. The image diameter of $f : D \to \mathbb{R}$, denoted $ImD(f)$, is $\max_{x \in D} f(x) - \min_{x \in D} f(x)$.

Theorem 1.1 (Lipschitz tester for hypercube)**.** *The Lipschitz property of functions $f : \mathcal{H}_d \to \delta\mathbb{Z}$ can be tested nonadaptively and with one-sided error in $O\left(\frac{d \cdot \min\{d, ImD(f)\}}{\delta\epsilon}\right)$ time for all $\delta \in (0, 1]$.*

For instance, if the range of f is $\{0, 1, 2\}$ then the tester runs in $O(d/\epsilon)$ time.

The tester first samples random points and checks if the image of the input function f, restricted to the samples, has appropriately small diameter for a Lipschitz function over $\mathcal{H}_d$ – namely, at most d. If f passes this test then it checks if the Lipschitz condition is satisfied for uniformly random edges of $\mathcal{H}_d$ and rejects if it finds a violation. To analyze the tester, we relate (in Lemma 3.1) the number of edges of $\mathcal{H}_d$ that are violated by a function to its distance to the Lipschitz property. The main tool in the analysis is the *averaging operator*, which we use to restore the Lipschitz property one dimension at a time. We build on ideas from [12], [10] which restored monotonicity one dimension at a time to analyze monotonicity tests for Boolean functions. Our averaging operator modifies values of f on the endpoints of each violated edge in a given dimension, bringing the two values sufficiently close. It can be thought of as computing an average of the values on the endpoints (however, one must be careful about how rounding to the nearest value in the range is done in order for our technique to work). One of the difficulties we overcome in the analysis is that the averaging operator might increase the number of violated edges in the previously restored dimensions. We introduce a potential function, called a *violation score*, that takes into account not only the number of violations, but also their magnitude. We prove that applying the averaging operator along one dimension does not increase the violation score in other dimensions. The main idea behind the proof is to break down the action of the averaging operator into small steps, captured by the *basic operator* which brings the endpoints of violated edges in a given dimension closer to each other by a small increment δ, and prove the desired statement for the basic operator.

The analysis of the tester in the proof of Theorem 1.1 does not apply directly to real-valued functions. By discretizing function values, in the full version, we obtain the following corollary for such functions.

Corollary 1.2. *There is an algorithm that gets parameters $\delta \in (0, 1], \epsilon \in (0, 1), d$ and oracle access to a function $f : \mathcal{H}_d \to \mathbb{R}$; it accepts if f is Lipschitz, rejects with probability at least 2/3 if f is ϵ-far from $(1 + \delta)$-Lipschitz and runs in $O\left(\frac{d \cdot \min\{d, ImD(f)\}}{\delta\epsilon}\right)$ time.*

We also give a lower bound on the query complexity of the tester for the hypercube which matches the upper bound in Theorem 1.1 for the case of the $\{0, 1, 2\}$ range and constant ϵ.

Theorem 1.3. *An (adaptive, two-sided error) tester of the Lipschitz property of functions $f : \mathcal{H}_d \to \mathbb{Z}$ must make $\Omega(d)$ queries. This holds even if the range of f is $\{0, 1, 2\}$.*

We prove Theorem 1.3 in the full version of this paper. We use the method presented in [5] of reducing a suitable communication complexity problem to the testing problem. [5] uses this method to prove (amongst other results) an $\Omega(d)$ lower bound for testing monotonicity of functions on $\{0, 1\}^d$ with a range of size $\Omega(\sqrt{d})$. Our lower bound for the Lipschitz property holds even for functions with a range of size 3.

Testing the Lipschitz property on the line: Next we give an efficient tester for the class of function properties which, in our terminology, are *edge-transitive* and *allow extension.* (Refer to the full version of this paper for the definition and discussion.) The Lipschitz property for functions on $f : \mathcal{L}_n \to R$ belongs to this class for most ranges R of interest. We characterize such ranges R as *discretely metrically convex* metric spaces. Metric convexity is a standard notion in geometric functional analysis (see, e.g., [2]). We define the discrete version, which is a weakening of the original notion in the following sense: all metrically convex spaces are also discretely metrically convex.

Definition 1.1 (Definition 1.3 of [2] and its relaxation)**.** *A metric space (R, d_R) is* metrically convex *(resp.,* discretely metrically convex*) if for all points $u, v \in R$ and positive real numbers (resp., positive integers) α and β satisfying $d_R(u, v) \le \alpha + \beta$, there exists $w \in R$ such that $d_R(u, w) \le \alpha$ and $d_R(w, v) \le \beta$.*

Our efficient tester for edge-transitive properties that allow extension, presented in the full version of this paper, builds on ideas from [4]. Specifically, for the Lipschitz property of functions $f : \mathcal{L}_n \to R$, it implies the following corollary.

Corollary 1.4. *The Lipschitz property of functions $f : \mathcal{L}_n \to R$ for every discretely metrically convex space R can be tested in time $O\left(\frac{\log n}{\epsilon}\right)$. In particular, the bound applies to the following metric spaces R: $(\mathbb{R}^k, \ell_p)$ for all $p \in [1, \infty)$, $(\mathbb{R}^k, \ell_\infty)$, $(\mathbb{Z}^k, \ell_1)$, $(\mathbb{Z}^k, \ell_\infty)$ and the shortest path metric d_G on all graphs G.*

The following theorem, proved in the full version of this paper, shows that the upper bound of Corollary 1.4 is tight for nonadaptive one-sided error testers. Even though it is stated for range $\mathbb{R}$ for concreteness, it trivially applies to $\mathbb{Z}^k$ and $\mathbb{R}^k$ for all k and metrics discussed above. (Note that it does not—and should not—apply to the shortest path metric on arbitrary graphs.)

Theorem 1.5. *A nonadaptive one-sided error tester of the Lipschitz property of functions $f : \mathcal{L}_n \to \mathbb{R}$ must make $\Omega(\log n)$ queries.*

To prove Theorem 1.5, we construct a family of $\Omega(\log n)$ functions which are $1/4$-far from Lipschitz and have pairwise disjoint sets of violated pairs. The construction has a clean description in terms of the *discrete derivative* function Δf, defined by $f(x) = \sum_{y \in [x]} \Delta f(y)$ for all $x \in [n]$.

Reconstruction of the Lipschitz property: We present a local filter of the Lipschitz property for functions of the form $f : [n]^d \to \mathbb{R}$ with lookup complexity $O((\log n + 1)^d)$. This result is stated in Theorem 1.6, which is proved in Section 4.

Theorem 1.6 (Local Lipschitz filters for Hypergrid)**.** *There is a deterministic nonadaptive local Lipschitz filter for functions $f : [n]^d \to \mathbb{R}$ with running time (and the number of lookups) $O((\log n + 1)^d)$ per query.*

We abstract the combinatorial object used in this filter as a *lookup graph* consistent with the domain graph. We show that the existence of a lookup graph implies a local Lipschitz filter where the lookup complexity of the filter is the maximum outdegree of a node in the lookup graph. We then obtain a lookup graph for $[n]$ with outdegree bounded by $O(\log n)$. Our construction builds on ideas of Ailon *et al.* [1] who gave a local monotonicity filter for functions $f : [n] \to \mathbb{R}$. We obtain a lookup graph for the hypergrid $\mathcal{H}_{n,d}$ by constructing a *strong* product of the lookup graphs for the line.

For functions of the form $\{0, 1\}^d \to \mathbb{R}$, we show that every nonadaptive reconstructor has lookup complexity exponential in d. The statement and the proof of the lower bound appear in full version of this paper. The main tool in the analysis is graph spanners, which were also used in [3] to prove lower bounds on local monotonicity reconstructors.

1.2. Applications

Our testers have applications to program analysis. Our filters have applications to data privacy.

Program Analysis: Certifying that a program computes a Lipschitz function has been studied in [7]. Applications described there include ensuring that a program is robust to noise in its inputs and ensuring that a program responds well to compiler optimizations that lead to an approximately equivalent program. For example, a Lipschitz function is guaranteed to respond proportionally to changes in input data (e.g., sensor measurements) due to rounding or other kinds of errors.

The methodology presented in [7] relies on inspecting the code of the program to verify that it computes a Lipschitz function. Their method might work for a particular program, but not apply to another functionally equivalent program with more complicated syntax. Efficient testers of the Lipschitz property allow one to approximately check if a program computes a Lipschitz function, while treating the program as a black box, without any syntactical restrictions. The only restriction we impose is on the domain and the range of the function computed by the program, since our tests are tailored to the domain and the range.

Data Privacy: The challenge in private data analysis is to release global statistics about the database while protecting the privacy of individual contributors. The database x can be modeled as a multiset (or a vector) over some domain U, where each element (resp., entry) $x_i \in U$ represents information contributed by one individual. One of main questions addressed in this area is: what information about x that does not heavily depend on individual entries x_i can we compute (and release) efficiently? There is a vast body of work on this problem in statistics and computer science, with [9] pioneering a line of work in cryptography. Subsequently, [11] defined a rigorous notion of privacy, called *differential privacy* (reviewed in Defition 5.1), and described the *Laplace mechanism* (reviewed in Theorem 5.1) for achieving differential privacy for releasing a given function f of the database x. The method is based on adding random noise from the Laplace distribution to $f(x)$, where the magnitude of the noise, i.e., the scale parameter of the distribution, is proportional to a Lipschitz constant of the function f.

Two major systems that release data while satisfying *differential privacy* have been implemented, both based on the Laplace mechanism. Both allow releasing functions of the database of the form $f : x \to \mathbb{R}$. In both implementations, the client sends a program to the server, requesting to evaluate it on the database, and receives the output of the program with Laplace noise added to it. However, the client is not trusted to provide a function with a low Lipschitz constant. The first approach relies on a language-based solution PINQ [17]. It imposes strict restrictions on the syntax of the programs that may be sent to the server holding the database, ensuring that programs evaluate Lipschitz functions. The second approach, taken in [18], allows

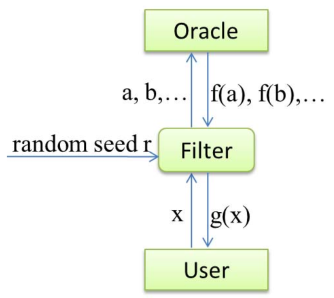


Figure 1. A filter

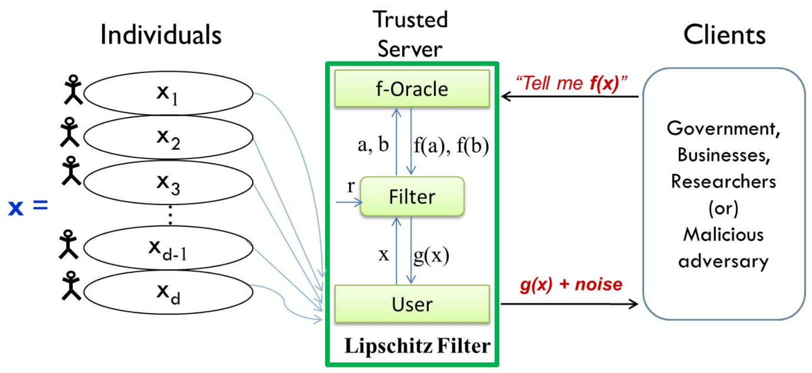


Figure 2. Use of a Lipschitz filter in private data analysis

for arbitrary programs. The privacy guarantee is ensured by enforcing that the program's output is always within its prespecified range. The range of the program must be declared and is used as a Lipschitz constant. Note that the range of a function can be much larger than its least Lipschitz constant. Therefore, the resulting mechanism may add overwhelming noise and destroy the information even when the function value could have been released privately with little noise.

The difficulty is that when f (supplied by a distrusted client) is given as a general-purpose program, it is hard to compute its least Lipschitz constant, or even an upper bound on it. Suppose we ask the client to supply a constant c such that f is c-Lipschitz. Unfortunately, as mentioned before, it is undecidable to even verify whether a function computed by a given Turing machine is c-Lipschitz for a fixed constant c. Applying the Laplace mechanism with c smaller than a Lipschitz constant (if the client is lying) would result in a privacy breach, while applying it with a generic upper bound on the least Lipschitz constant of f would result in overwhelming noise.

In Section 5.1, we describe and analyze a different solution, which we call the *filter mechanism*, that can be used to release a function f when a Lipschitz constant of f is provided by a distrusted client. (See Figure 2.) The filter mechanism is differentially private and adds the same amount of noise as the Laplace mechanism for an honest client. Instead of directly running a program f, provided by the client, on the database x, the server calls a local Lipschitz filter on query x with f as an oracle. The filter outputs $g(x)$ instead of $f(x)$, where g is Lipschitz[2]. Crucially, since the filter is local, it guarantees that g does not depend on the database x. That is, the client could have computed g by herself, based on f. Consequently, releasing $g(x)$ via the Laplace mechanism is differentially private. Moreover, if the client is honest and provides a program that computes a Lipschitz function f, the output function g of the filter is identical to f. In this case, the noise added to the answer is identical to that of the Laplace mechanism[3].

Let $Lap(\lambda)$ denote the Laplace distribution on $\mathbb{R}$ with the *scale parameter* λ. Its density function is $f_\lambda(y) = \frac{1}{2\lambda}e^{-\frac{|y|}{\lambda}}$. The following theorem, proved in Section 5.1, summarizes the performance of the filter mechanism.

Theorem 1.7 (Filter Mechanism)**.** *Fix $c, \epsilon > 0$. Let A be a local Lipschitz filter of functions $f : D \to \mathbb{R}^t$ where the Lipschitz property is with respect to (D, d_D). For all functions $f : D \to \mathbb{R}^t$, the following algorithm (which receives f as an oracle) is ϵ-differentially private: $\mathcal{A}^f_{Fil}(x) = c \cdot A(\frac{1}{c} \cdot f, x) + (Y_1, \ldots, Y_t)$, where $Y_i \overset{i.i.d.}{\sim} Lap(c/\epsilon)$ for all $i \in [t]$.*

Moreover, for all c-Lipschitz functions f, the outputs of the filter and Laplace mechanisms are identical with probability at least $1-\delta$, where δ is the error probability of the local filter.

In Section 5.2, we instantiate the filter mechanism with our filter from Theorem 1.6 to obtain an efficient private algorithm for releasing functions $f : x \to \mathbb{R}$ of the databases x which can be represented as multisets and for which an upper bound on the multiplicity of

[2]If one needs to ensure that a function is c-Lipschitz, the function can be rescaled.

[3]We do not insist that f and g differ only on a small number of points, since we call our filter only on one database x. If we did, a dishonest client would be penalized for fewer instances of x. Observe that the amount of distortion reconstruction introduces by substituting $f(x)$ with $g(x)$ does not depend on the distance of f to the Lipschitz property: it could be Lipschitz everywhere, besides x, but $f(x)$ would be changed anyway. However, it is not hard to see that our filter never changes $f(x)$ by more than $\max_y \{|f(y) - f(x)| + d_G(x, y)\}$.

all elements of the universe U is known. (Note that the number of people in databases is a trivial upper bound.) When the client provides a correct Lipschitz constant, the resulting filter mechanism has the same expected error as the Laplace mechanism. Our mechanism is differentially private even for dishonest clients.

We show that when no reliable Lipschitz constant of f is given, previously known differentially private mechanisms (specifically, those based on the Laplace mechanism) either have a substantially higher running time (because they verify the Lipschitz constant by brute force) or have a higher expected error for a large class of functions f. Specifically, suppose that U has size k, that is, the individuals can have one of k *types*, and consider functions f that compute the number of individuals of types $S \subseteq [k]$ for $|S| = \Omega(k)$. We show that the *noisy histogram* approach (based on the Laplace mechanism) incurs an expected $\Omega(\sqrt{k})$ error in answering the query. In contrast, our filter mechanism has expected error $O(1/\epsilon)$ while preserving differential privacy even in the presence of distrusted clients. The following theorem, proved in the full version, summarizes the comparison of the filter mechanism to the noisy histogram approach.

Theorem 1.8. *For some functions f, releasing f results in expected error $\Omega(\sqrt{k}/\epsilon)$ with the noisy histogram approach, but only $O(1/\epsilon)$ with the filter mechanism.*

2. Preliminaries

Testing Properties of functions: Given functions f, g on the same domain D, the *distance* between f and g, denoted $Dist(f, g)$, is the number of points in D on which f and g differ. We say f is ϵ-far from a property $\mathcal{P}$ if $Dist(f, g) \geq \epsilon \cdot |D|$ for all functions g satisfying $\mathcal{P}$. A (*two-sided* error, *adaptive*) *q-query* tester for a property $\mathcal{P}$ is a randomized algorithm, which given oracle access to a function f and a parameter $\epsilon \in (0, 1)$ makes at most q queries to the oracle f and can distinguish, with probability 2/3, the case that f satisfies $\mathcal{P}$ from the case that f is ϵ-far from $\mathcal{P}$. A tester has *one-sided error* if it always accepts functions satisfying $\mathcal{P}$. It is *nonadaptive* if the queries to f do not depend on the answers to the previous queries.

Local Property Reconstruction: Local reconstruction was defined in [20]. The variant we consider is from [3].

Definition 2.1 (Local filter). *A* local filter *for reconstructing property $\mathcal{P}$ is an algorithm A that has oracle access to a function $f : D \to R$ and to an auxiliary random string ρ (the "random seed"), and takes as input $x \in D$. For fixed f and ρ, A runs deterministically on input x to produce an output $A_{f,\rho}(x) \in R$. The function $g(x) = A_{f,\rho}(x)$ output by the filter must satisfy $\mathcal{P}$ for all f and ρ. In addition, if f satisfies $\mathcal{P}$ then g must be identical to f with probability at least $1 - \delta$ for some* error probability $\delta \leq 1/3$, *where the probability is taken over ρ.*

When answering a query $x \in D$, a filter may access values of f at domain points of its choice using its oracle. Each accessed domain point is called a *lookup* to distinguish it from the client query x. A local filter is *nonadaptive* if its lookups on input query x do not depend on answers given by the oracle.

3. Testing the Lipschitz Property

In this section, we show how to test if a function $f : \mathcal{H}_d \to \delta\mathbb{Z}$ is Lipschitz and explain the main ideas from the proof of Theorem 1.1.

W.l.o.g., we assume that that $1/\delta$ is an integer.

Observe that a function is Lipschitz if its values on the endpoints of every edge differ by at most 1. An edge $\{x, y\}$ is *violated* if $|f(x) - f(y)| > 1$. Since $\mathcal{H}_d$ has diameter d, no values in the image of a Lipschitz function should differ by more than d.

Definition 3.1 (Image diameter). *The* image diameter *of a function $f : D \to \mathbb{R}$, denoted $ImD(f)$, is the difference between the maximum and the minimum values attained by f, i.e., $\max_{x\in D} f(x) - \min_{x\in D} f(x)$.*

First, our algorithm approximates the image diameter of the input function f by computing the image diameter of f, restricted to random samples, and rejects if it is greater than d. Then it looks for violations by sampling hypercube edges uniformly at random.

```
LIPSCHITZ-TEST($f : \{0,1\}^d \to \delta\mathbb{Z}, d, \delta, \epsilon$)
1  Select $t = \lceil 12/\epsilon \rceil$ vertices $z_1, \ldots, z_t$ uniformly and
      independently at random from the hypercube $\mathcal{H}_d$.
2  Let $r = \max_{i=1}^{t} f(z_i) - \min_{i=1}^{t} f(z_i)$.
3  if $r > d$, reject.
4  Select $\lceil (2 \cdot dr)/\delta\epsilon \rceil$ edges uniformly and
      independently at random from the hypercube $\mathcal{H}_d$.
5  if any of the selected edges are violated, reject;
      otherwise, accept.
```

The main tool in the analysis of our test is Lemma 3.1, proved in Sections 3.1–3.2. In the full version, we derive Theorem 1.1 from this lemma.

Lemma 3.1 (Main). *Let function $f : \{0, 1\}^d \to \delta\mathbb{Z}$ be ϵ-far from Lipschitz. Let $V(f)$ denote the number of edges of $\mathcal{H}_d$ violated by f. Then $V(f) \geq \frac{\delta\epsilon \cdot 2^{d-1}}{ImD(f)}$.*

3.1. Averaging Operator A_i

To prove Lemma 3.1, we show how to transform an arbitrary function $f : \{0, 1\}^d \to \delta\mathbb{Z}$ into a Lipschitz function by changing f on a set of points, whose size is related to the number of the hypercube edges violated by f. This is achieved by repairing one dimension of the hypercube $\mathcal{H}_d$ at a time with the averaging operator A_i, defined below. The operator modifies values of f on the endpoints of each violated edge in dimension i, bringing the two values sufficiently close. It can be thought of as computing an average of the values on the endpoints and rounding it down and up to the closest values in $\delta\mathbb{Z}$ to obtain new assignments for the endpoints. Let $\lfloor x \rfloor_\delta$ (resp., $\lceil x \rceil_\delta$) be the smallest (resp., largest) value in $\delta\mathbb{Z}$ not greater (resp., not smaller) than x.

Definition 3.2 (Averaging operator A_i)**.** *Given $f : \{0,1\}^d \to \delta\mathbb{Z}$, for each violated edge $\{x, y\}$ along dimension i, where vertex names x and y are chosen so that $f(x) < f(y) - 1$, define $A_i[f](x) = \left\lfloor \frac{f(x)+f(y)}{2} \right\rfloor_\delta$ and $A_i[f](y) = \left\lceil \frac{f(x)+f(y)}{2} \right\rceil_\delta$.*

We would like to argue that while we are repairing dimension i with the averaging operator, other dimensions are not getting worse. Unfortunately, the number of violated edges along other dimensions can increase. Instead, we keep track of our progress by looking at a different measure, called the *violation score*.

Definition 3.3 (Violation score)**.** *The* violation score *of an edge $\{x, y\}$ with respect to function f, denoted $\mathbf{vs}(\{x, y\})$, is $\max(0, |f(x) - f(y)| - 1)$. The* violation score *of dimension i, denoted $VS^i(f)$, is the sum of violation scores of all edges along dimension i.*

The violation score of an edge is positive iff the edge is violated. Moreover, the violation score of a violated edge with respect to a $\delta\mathbb{Z}$-valued function is contained in the interval $[\delta, \mathit{ImD}(f)]$. Let $V^i(f)$ be the number of edges along dimension i violated by f. Then

$$\delta V^i(f) \le VS^i(f) \le V^i(f) \cdot \mathit{ImD}(f). \qquad (1)$$

Later, we use (1) to bound the number of values of f modified by A_i in terms of $V^i(f)$. Next lemma shows that A_i does not increase the violation score in dimensions other than i.

Lemma 3.2. *For all $i, j \in [d]$, where $i \neq j$, and every function $f : \{0,1\}^d \to \delta\mathbb{Z}$, applying the* averaging *operator A_i does not increase the violation score in dimension j, i.e., $VS_j(A_i[f]) \le VS_j(f)$.*

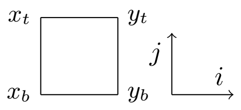
Proof. The main idea behind the proof is to break down the action of the averaging operator A_i into small steps and prove that each step along dimension i does not increase the violation score in dimension j. Each small step is captured by the *basic* operator B_i, defined next.

Definition 3.4 (Basic operator B_i)**.** *Given $f : \{0,1\}^d \to \delta\mathbb{Z}$, for each violated edge $\{x, y\}$ along dimension i, where vertex names x and y are chosen so that $f(x) < f(y) - 1$, define $B_i[f](x) = f(x) + \delta$ and $B_i[f](y) = f(y) - \delta$.*

It is easy to see that applying A_i is equivalent to applying B_i multiple times until no edges along dimension i are violated. Therefore, it is enough prove Lemma 3.2 for B_i instead of A_i.

Note that the edges along dimensions i and j form disjoint squares in the hypercube. Therefore, the special case of Lemma 3.2 for f restricted to each of these squares individually (where each such restriction is a two-dimensional function) allows us to prove the lemma for dimensions i and j by summing the inequalities over all such squares. It remains to prove the lemma for $d = 2$ and B_i instead of A_i.

Note that we may assume w.l.o.g. that $1/\delta$ is an integer. This is because f is Lipschitz iff f/δ is $1/\delta$-Lipschitz. Since f/δ is an integer-valued function, it is $1/\delta$-Lipschitz iff it is $\lfloor 1/\delta \rfloor$-Lipschitz. Let $c = \lfloor 1/\delta \rfloor$ and $f' = f/(\delta \cdot c)$. Then f' is Lipschitz iff f is Lipschitz. Therefore, testing if $f : \mathcal{H}_d \to \delta\mathbb{Z}$ is Lipschitz is equivalent to testing if $f' : \mathcal{H}_d \to (1/c)\mathbb{Z}$ is Lipschitz for the integer c defined above.



Consider a two-dimensional function $f : \{x_t, x_b, y_t, y_b\} \to \delta\mathbb{Z}$ with vertices x_t, x_b, y_t, y_b positioned as depicted. We show that an application of the basic operator B_i along the horizontal dimension does not increase the violation score of the vertical dimension. If the violation scores of the vertical edges do not increase, the proof is complete. Assume w.l.o.g. the violation score of the left vertical edge $\{x_t, x_b\}$ increases. Also w.l.o.g. assume $B_i[f](x_t) > B_i[f](x_b)$ (otherwise, we can swap the horizontal edges on our picture.) Then B_i increases $f(x_t)$ and/or decreases $f(x_b)$. Assume w.l.o.g. B_i increases $f(x_t)$. (The case when B_i decreases $f(x_b)$ is symmetrical). Then $\{x_t, y_t\}$ is violated with $f(x_t) < f(y_t)$. Moreover, since f is a $\delta\mathbb{Z}$-valued function and $1/\delta$ is an integer, $f(y_t) \ge f(x_t) + 1 + \delta$. The application of the basic operator increases $f(x_t)$ by δ and decreases $f(y_t)$ by δ.

If the bottom edge is not violated then $f(x_b) \ge f(y_b) - 1$ and the basic operator does not change $f(x_b)$ and $f(y_b)$. Since $\mathbf{vs}(\{x_t, x_b\})$ increases, $f(x_t) > f(x_b) + 1 - \delta$. Integrality of $1/\delta$ implies $f(x_t) \ge f(x_b) + 1$. Combining the three inequalities derived so far, we get $f(y_t) \ge f(x_t) + 1 + \delta \ge f(x_b) + 2 + \delta \ge f(y_b) + 1 + \delta$. Thus, $\mathbf{vs}(\{x_t, x_b\})$ increases by δ, while $\mathbf{vs}(\{y_t, y_b\})$ decreases by δ, keeping the violation score along the vertical dimension unchanged.

If the bottom edge is violated then, since $\mathbf{vs}(\{x_t, x_b\})$ increases and $1/\delta$ is integral, $f(x_t) \geq f(x_b) + 1 - \delta$. Also, $f(x_b)$ must decrease, implying $f(x_b) > f(y_b)+1$. Therefore, $f(y_t) \geq f(x_t) + 1 + \delta \geq f(x_b) + 2 > f(y_b) + 3$. Recall that $\delta \leq 1$. Thus, $\mathbf{vs}(\{x_t, x_b\})$ increases by at most 2δ, while $\mathbf{vs}(\{y_t, y_b\})$ decreases by 2δ, ensuring that the violation score along the vertical dimension does not increase. □

3.2. *Proof of Lemma 3.1*

The crux of the proof is showing how to make a function $f : \{0,1\}^d \to \delta\mathbb{Z}$ Lipschitz by redefining it on at most $\frac{2}{\delta} \cdot V(f) \cdot \mathit{ImD}(f)$ points. We apply a sequence of averaging operators as follows: we define $f_0 = f$ and for all $i \in [d]$, let $f_i = A_i[f_{i-1}]$.

$$f = f_0 \xrightarrow{A_1} f_1 \xrightarrow{A_2} f_2 \to \cdots \to f_{d-1} \xrightarrow{A_d} f_d.$$

We claim that f_d is Lipschitz. By Definition 3.2, each step above makes one dimension i free of violated edges. Recall that the violation score VS^i is 0 iff dimension i has no violated edges. Therefore, by Lemma 3.2, A_i preserves the Lipschitz property along dimensions fixed in the previous steps. Thus, eventually there are no violated edges, and f_d is Lipschitz.

Now we bound the number of points on which f and f_d differ, that is, $\mathit{Dist}(f, f_d)$. For all $i \in [d]$,

$$\mathit{Dist}(f_{i-1}, f_i) = \mathit{Dist}(f_{i-1}, A_i[f_{i-1}]) \leq 2 \cdot V^i(f_{i-1})$$
$$\leq \frac{2}{\delta} \cdot VS^i(f_{i-1}) \leq \frac{2}{\delta} \cdot VS^i(f) \leq \frac{2}{\delta} \cdot V^i(f) \cdot \mathit{ImD}(f).$$

The first inequality holds because A_i modifies f only on the endpoints of violated edges along dimension i. The second and the fourth inequality follow from (1). The third inequality holds because, by Lemma 3.2, the operators A_j for $j \neq i$ do not increase the violation score in dimension i. By the triangle inequality, the distance from f to f_d is

$$\mathit{Dist}(f, f_d) \leq \sum_{i\in[d]} \mathit{Dist}(f_{i-1}, f_i)$$
$$\leq \sum_{i\in[d]} \frac{2}{\delta} \cdot V^i(f) \cdot \mathit{ImD}(f) = \frac{2}{\delta} \cdot V(f) \cdot \mathit{ImD}(f). \quad (2)$$

Consider a function f which is ϵ-far from the Lipschitz property. Since f_d is Lipschitz, $\mathit{Dist}(f, f_d) \geq \epsilon 2^d$. Using (2), we get $V(f) \geq \frac{\epsilon\delta\cdot 2^{d-1}}{\mathit{ImD}(f)}$, as required.

4. Reconstructing the Lipschitz property

In this section, we prove Theorems 1.6, giving local filters of the Lipschitz property for functions $f : \mathcal{L}_n \to \mathbb{R}$ and $f : \mathcal{H}_{n,d} \to \mathbb{R}$. Our filters are deterministic and nonadaptive. We abstract the combinatorial object used in these filters as a *lookup graph* consistent with the domain graph. We start by defining lookup graphs in Definition 4.2. In Lemma 4.1, we show how to use them to construct Lipschitz filters. Finally, we construct lookup graphs for the line and the hypergrid in Lemma 4.3. Lemmas 4.1 and 4.3 imply Theorem 1.6.

Definition 4.1 (Out-neighbors, outdegree)**.** *Given a directed graph $H = (V, E_H)$ and a node $u \in V$, let $\mathcal{N}_H(u)$ be the set $\{z \in V \mid (u,z) \in E_H\}$ of out-neighbors of u in H. Let $\mathcal{N}^*_H(u) = \mathcal{N}_H(u) \cup \{u\}$. (We omit the subscript H when the graph is clear from the context.) We denote the maximum outdegree of a node in H by $outdegree(H)$.*

Definition 4.2 (Lookup graph)**.** *Given an undirected graph $G = (V, E)$, a* lookup graph *of G is a directed graph $H = (V, E_H)$ satisfying the following properties:*

- *Consistency: for all $x, y \in V$, some $z \in \mathcal{N}^*_H(x) \cap \mathcal{N}^*_H(y)$ is on a shortest path between x and y in G.*
- *(Strict) Containment: $(x,y) \in E_H \Rightarrow \mathcal{N}_H(y) \subset \mathcal{N}_H(x)$.*

Lemma 4.1. *If a graph G has a lookup graph H then there is a nonadaptive local Lipschitz filter for real-valued functions on G with lookup complexity $outdegree(H)$ and running time $O(outdegree(H))$.*

Proof: We describe a filter which receives a lookup graph H and $f : V(H) \to \mathbb{R}$ as inputs. We assume that the filter has access to the domain graph G and that distances in G can be computed in constant time.

We say a function $f : D \to R$ is *Lipschitz on $D' \subseteq D$* if for all pairs $(x,y) \in D' \times D'$, the Lipschitz condition is satisfied, namely, $d_R(f(x), f(y)) \leq d_D(x,y)$.

```
FILTER_H(f, x)
1  if $\mathcal{N}(x)$ is empty, output $g(x) = f(x)$;
2  for each vertex $z$ in $\mathcal{N}(x)$,
           recursively compute $g(z) = \text{FILTER}_H(f, z)$;
3  if setting $g(x) = f(x)$ makes $g$ Lipschitz on $\mathcal{N}^*(x)$
4      then output $g(x) = f(x)$
5      else output $g(x) = \max_{z\in\mathcal{N}(x)} (g(z) - d_G(x,z))$.
```

We proceed to prove correctness of the filter. The recursion on Line 2 terminates because H satisfies the *containment* property and, consequently, the size of the out-neighbor set decreases strictly with every successive recursive call.

Claim 4.2. *If function f is Lipschitz on $\mathcal{N}^*(x)$ and on $\mathcal{N}^*(y)$, it is also Lipschitz on $\{x, y\}$.*

Proof: Let $z \in \mathcal{N}^*(x) \cap \mathcal{N}^*(y)$ be a vertex which lies on a shortest path between x and y in G (guaranteed to exist by the *consistency* property of H). From the statement of the claim, f is Lipschitz on $\{x, z\}$ and $\{y, z\}$. Since z lies on a shortest path between x and y in G, function f is Lipschitz on $\{x, y\}$. ■

Using Claim 4.2, it is sufficient to prove that for each $x \in V$, g is Lipschitz on $\mathcal{N}^*(x)$. The proof is by strong induction on $|\mathcal{N}(x)|$. The base case (when $|\mathcal{N}(x)| = 0$) holds for trivial reasons. For the inductive case, let $|\mathcal{N}(x)| = k > 0$. Since each $z \in \mathcal{N}(x)$ has $|\mathcal{N}(z)| < k$, we may assume (by the induction hypothesis) g is Lipschitz on $\mathcal{N}^*(z)$ for all $z \in \mathcal{N}(x)$. Then Claim 4.2 implies that g is Lipschitz on $\mathcal{N}(x)$. The fact that g is Lipschitz on $\mathcal{N}^*(x)$ then follows from statements on lines 3 and 4.

On query x, the filter only looks up out-neighbors of x because of the *containment* property of H. Therefore, the lookup complexity of the filter is at most $outdegree(H)$. Moreover, if the filter stores the value of $g(z)$ the first time it is computed and reuses it later, the running time is $O(outdegree(H))$. ■

Lemma 4.3 (Lookup graph constructions)**.** *The line graph $\mathcal{L}_n$ has a lookup graph H with $outdegree(H) = O(\log n)$. The hypergrid $\mathcal{H}_{n,d}$ has a lookup graph H with $outdegree(H) = (1 + \log n)^d$.*

Proof: To construct a lookup graph for the line $\mathcal{L}_n$, consider a balanced (rooted) *binary search tree* T on the set $V_n = [n]$. Recall that the lowest common ancestor (LCA) of a pair of vertices x, y in T is a common ancestor of x and y which is furthest from the root. We now construct H as follows: For each $x \in [n]$, if $w \neq x$ is an ancestor of x in T, we add an edge (x, w) in H. Now, for distinct $x, y \in [n]$, the vertex $LCA(x, y)$ is common to both out-neighbors of x and y. Moreover, the binary search tree property implies that it lies on the shortest path between x and y: for all $x < y$, $x \leq LCA(x, y) \leq y$. This verifies that H is a lookup graph of $\mathcal{L}_n$. From definition of H, it is also clear that H satisfies the *containment* property. Finally, $outdegree(H) = O(\log n)$.

To construct a lookup graph for $\mathcal{H}_{n,d}$, we use a strong product of lookup graphs for the line; details are provided in the full version of this paper. ■

Theorem 1.6 follows from Lemmas 4.1 and 4.3.

5. Application to Data Privacy

In Section 5.1, we review differential privacy and the Laplace mechanism from [11] and describe our filter mechanism. In Section 5.2, we instantiate the filter mechanism with the filter from Theorem 1.6 to obtain a private and efficient algorithm for releasing functions f of the data when a Lipschitz constant of the function is provided by a distrusted client.

5.1. Filter Mechanism

There are several ways to model a database. It can be represented as a vector or a multiset where each component (or element) represents an individual's data and takes values in some fixed universe U. In the latter case, equivalently, it can be represented by a *histogram* – that is, a vector where the ith component represents the number of times the ith element of U occurs in the database. Two databases x and y are *neighbors* if they differ in one individual's data. For example, if x and y are histograms, they are neighbors if they differ by 1 in exactly 1 component. The results of this section apply to all of these models. Let D denote the set of all databases x. The notion of neighboring databases induces a metric d_D on D such that $d_D(x, y) = 1$ iff x and y are neighbors.

Definition 5.1 (Differential privacy, [11])**.** *Fix $\epsilon > 0$. A randomized algorithm $\mathcal{A}$ is ϵ-differentially private if for all neighbors $x, y \in D$, and for all subsets S of outputs,* $\Pr[\mathcal{A}(x) \in S] \leq e^\epsilon \Pr[\mathcal{A}(y) \in S]$.

Recall that $Lap(\lambda)$ denote the Laplace distribution on $\mathbb{R}$ with the *scale parameter* λ. The *Laplace mechanism* [11] is a randomized algorithm for evaluating functions on databases privately and efficiently.

Theorem 5.1 (Laplace Mechanism [11])**.** *Fix $c, \epsilon > 0$. For all functions $f : D \to \mathbb{R}^t$ which are c-Lipschitz on the metric space (D, d_D), the following algorithm (which receives f as an oracle) is ϵ-differentially private: $\mathcal{A}^f_{Lap}(x) = f(x) + (Y_1, \ldots, Y_t)$, where $Y_i \overset{i.i.d.}{\sim} Lap(c/\epsilon)$ for all $i \in [t]$.*

Next we prove Theorem 1.7 (from Section 1.2) that summarizes the performance of our *filter mechanism.*

Proof of Theorem 1.7: Let x be the input database and g_ρ the output function of the local Lipschitz filter A with random seed fixed to ρ. Since the filter is local, g_ρ is well defined on D. In particular, this means that g_ρ can be computed by the user without the knowledge of x and therefore does not disclose anything about the database x. Moreover, g_ρ is guaranteed to be 1-Lipschitz and therefore, $c \cdot g_\rho$ is c-Lipschitz. The filter mechanism can thus be seen as an application of the Laplace mechanism on the c-Lipschitz function $c \cdot g_\rho$. By Theorem 5.1, the algorithm $\mathcal{A}^f_{Fil}$ is ϵ-differentially private. Since ρ was arbitrary, above analysis holds for any choice of ρ, i.e., any instantiation of the filter A.

For the second part of the theorem, note that if f is c-Lipschitz, the function that filter A gets as an input oracle, $\frac{1}{c} \cdot f$, is Lipschitz. Thus, the output function of the filter is identical to its input function with probability at least $1 - \delta$. Since the output of the filter is scaled by c, the second part of the theorem follows. ■

5.2. Filter Mechanism for Histograms

Theorem 1.7 applies to arbitrary metric spaces (D, d_D). In this section, we instantiate it with the local Lipschitz filter for functions from the hypergrid to real numbers, described in Theorem 1.6, and analyze its performance.

Recall that each individual's data is an element of an arbitrary domain U. Suppose that U consists of k elements, that is, the individuals can have one of k *types*. In this section, we model a database x as a histogram, i.e., a vector in $\mathbb{R}^k$, where the ith component represents the number of times the ith element of U occurs in the database. Consider the set of databases which contain at most m individuals of each type. The corresponding set of histograms is $D = \{0, ..., m\}^k$. Recall that two histograms are neighbors if they differ by 1 in exactly one of the components. In this case, we can identify the metric space (D, d_D) with the hypergrid $\mathcal{H}_{m+1,k}$ (with the convention that vertices are vectors with entries in $\{0, ..., m\}$ instead of $[m + 1]$). Therefore, we can use our local Lipschitz filter from Theorem 1.6 in the filter mechanism to release functions $f : D \to \mathbb{R}$. The performance of the resulting algorithm is summarized in Corollary 5.2. We also bound the *error* of the mechanism. Given a function $f : D \to \mathbb{R}$ and a (randomized) mechanism A for evaluating f, let $\mathcal{E}(f, A) = \sup_{x \in D} \mathbb{E}[|A(x) - f(x)|]$ be the *error* of the mechanism A in computing f.

Corollary 5.2 (Filter mechanism for histograms)**.** *Fix $c, \epsilon > 0$. For all functions $f : D \to \mathbb{R}$, the filter mechanism of Theorem 1.7 instantiated with the local filter of Theorem 1.6 is ϵ-differentially private and its running time is bounded by $(\log(m + 1) + 1)^k$ evaluations of f. In addition, for c-Lipschitz functions f on D, the error of the mechanism, $\mathcal{E}(f, A_{Fil})$ is $O(c/\epsilon)$.*

The proof of the corollary appears in the full version.

Acknowledgment

The authors would like to thank A. Smith and G. Yaroslavtsev for useful comments and discussions.

References

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu, "Property-preserving data reconstruction," *Algorithmica*, vol. 51, no. 2, pp. 160–182, 2008.

[2] Y. Benyamini and J. Lindenstrauss, *Geometric nonlinear functional analysis. Vol. 1*, ser. Colloquium Publications. Amer. Math. Soc., 2000, vol. 48.

[3] A. Bhattacharyya, E. Grigorescu, M. Jha, K. Jung, S. Raskhodnikova, and D. P. Woodruff, "Lower bounds for local monotonicity reconstruction from transitive-closure spanners," in *RANDOM*, 2010, pp. 448–461.

[4] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. P. Woodruff, "Transitive-closure spanners," in *SODA*. ACM-SIAM, 2009, pp. 932–941.

[5] E. Blais, J. Brody, and K. Matulef, "Property testing lower bounds via communication complexity," in *CCC*. IEEE Computer Society, 2011, pp. 210–220.

[6] S. Chakraborty, E. Fischer, and A. Matsliah, "Query complexity lower bounds for reconstruction of codes," in *Symposium on Innovations in Computer Science*, 2011.

[7] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour, "Proving programs robust," in *ESEC/FSE*, 2011, to appear.

[8] B. Chazelle and C. Seshadhri, "Online geometric reconstruction," in *SoCG*. ACM, 2006, pp. 386–394.

[9] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in *PODS*, 2003, pp. 202–210.

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

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

[12] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky, "Testing monotonicity." *Combinatorica*, vol. 20, no. 3, pp. 301–337, 2000.

[13] O. Goldreich, S. Goldwasser, and D. Ron, "Property testing and its connection to learning and approximation," *J. ACM*, vol. 45, no. 4, pp. 653–750, 1998.

[14] M. Jha and S. Raskhodnikova, "Testing and reconstruction of Lipschitz functions with applications to data privacy," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. TR11-057, 2011.

[15] S. Kale, Y. Peres, and C. Seshadhri, "Noise tolerance of expanders and sublinear expander reconstruction," in *FOCS*. IEEE Computer Society, 2008, pp. 719–728.

[16] C. McDiarmid, "On the method of bounded differences," in *Surveys in Combinatorics*. Cambridge University Press, 1989, pp. 148–188.

[17] F. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," *Commun. ACM*, vol. 53, no. 9, pp. 89–97, 2010.

[18] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel, "Airavat: Security and privacy for mapreduce," in *NSDI*, 2010, pp. 297–312.

[19] R. Rubinfeld and M. Sudan, "Robust characterization of polynomials with applications to program testing," *SIAM J. Comput.*, vol. 25, no. 2, pp. 252–271, 1996.

[20] M. E. Saks and C. Seshadhri, "Local monotonicity reconstruction," *SIAM J. Comput.*, vol. 39, no. 7, pp. 2897–2926, 2010.