Electronic Colloquium on Computational Complexity, Report No. 57 (2011)

Testing and Reconstruction of Lipschitz Functions with Applications to Data Privacy

Madhav Jha* Sofya Raskhodnikova*

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 functions 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.) In other words, Lipschitz functions are not very sensitive to small changes in the input.

We initiate the study of testing and local reconstruction of the Lipschitz property of functions. A property *tester* 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 an arbitrary function f to ensure that the reconstructed function g has the desired property (in this case, is Lipschitz), changing f only when necessary. A local filter is given a function f and a query x and, after looking up the value of f on a small number of points, it has to output $g(x)$ for some function g, which has the desired property and does not depend on x. If f has the property, g must be equal to f.

We consider functions over domains $\{0,1\}^d$, $\{1, \dots, n\}$ and $\{1, \dots, n\}^d$, equipped with ℓ_1 distance. We design efficient testers of the Lipschitz property for functions of the form $f : \{0,1\}^d \to \delta\mathbb{Z}$, 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. In the first case, the tester runs in time $O(d \cdot \min\{d, r\} / \delta\epsilon)$, where r is the diameter of the image of f; in the second, in time $O((\log n)/\epsilon)$. We give corresponding lower bounds of $\Omega(d)$ and $\Omega(\log n)$ on the query complexity (in the second case, only for nonadaptive 1-sided error testers). Our lower bound for functions over $\{0,1\}^d$ is tight for the case of the $\{0, 1, 2\}$ range and constant ϵ. The first tester implies an algorithm for functions of the form $f : \{0,1\}^d \to \mathbb{R}$ that distinguishes Lipschitz functions from functions that are ϵ-far from $(1 + \delta)$-Lipschitz. We also present a local filter of the Lipschitz property for functions of the form $f : \{1, \dots, n\}^d \to \mathbb{R}$ with lookup complexity $O((\log n + 1)^d)$. For functions of the form $\{0,1\}^d$, we show that every nonadaptive local filter has lookup complexity exponential in d.

The testers that we developed have applications to programs analysis. The reconstructors have applications to data privacy. For the first application, the Lipschitz property of the function computed by a program corresponds to a notion of robustness to noise in the data. 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 Smith, TCC 2006). We give a differentially private mechanism, based on local filters, for releasing a function f when a 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 either have a substantially higher running time or have a higher expected error for a large class of symmetric functions f.

*Pennsylvania State University, USA. Email: {mxj201,sofya}@cse.psu.edu. Supported by National Science Foundation (NSF/CCF CAREER award 0845701).

ISSN 1433-8092

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 $1/c$ converts a c-Lipschitz function into a Lipschitz function.) In other words, Lipschitz functions are not very sensitive to small changes in the 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 [33]; in program analysis, it is considered as a measure of robustness to noise [18]; in data privacy, it is used to scale noise added to output $f(x)$ to preserve differential privacy of a database x [23]. 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* [42, 28] is given oracle access to an object (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 [3] is beneficial when an algorithm, call it A, is computing on a large dataset and the algorithms'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). Thus, A is computing with reconstructed data g instead of its original input f.

Local reconstruction [43] 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 [10], 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.

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

Previous work on property testing and reconstruction. Property testing [28, 42] 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 (see [39, 41] for recent surveys), most of which are not directly related to the problems we consider here. A notable exception is a line of work on testing monotonicity of functions [24, 27, 22, 7, 26, 25, 30, 1, 11, 10, 14, 36, 2] which has provided several techniques that are surprisingly useful for testing the Lipschitz property. We discuss this connection between monotonicity and the Lipschitz property in Section 1.1.

Property preserving reconstruction [3] has been studied for monotonicity of functions [3, 43, 10], convexity of points [19], graph expansion [31] and error-correcting codes [15]. The local model is addressed in [43, 15, 10], with only [43] 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 $\{\{x, y\} : \exists \text{ unique } i \in [d] \text{ such that } |y_i - x_i| = 1 \text{ and for } j \neq i, y_j = x_j\}$. 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 $\{1,2\}^d$.

Relationship to monotonicity of functions. A function $f : G \to R$, where G is a partially ordered set equipped with partial order $\prec$ (equivalently, a directed acyclic graph) and R is a linear order, is *monotone* if $f(x) \leq f(y)$ for all $x \prec y$ (equivalently, all edges (x, y) in G). Testing and reconstruction of monotone functions has been extensively studied, with a particular focus on functions over directed hypergrids of different sizes and dimensions. In particular, the directed line $\overrightarrow{\mathcal{L}_n}$ was studied in [24, 22, 11, 25], the directed hypercube $\overrightarrow{\mathcal{H}}_d$ in [27, 22, 26, 14] and the directed hypergrid $\overrightarrow{\mathcal{H}}_{n,d}$ in [27, 22, 7, 30, 10, 2, 43], where the directed hypergrids are obtained from corresponding undirected hypergrids by orienting their edges according to the standard partial order, $\prec$, on the hypergrids: for distinct vertices $x, y \in [n]^d$, $x \prec y$ iff for all $i \in [d]$, $x_i \leq y_i$. (Specifically, $\overrightarrow{\mathcal{H}}_{n,d}$ has (x, y) as an edge iff $\{x, y\}$ is an edge in $\mathcal{H}_{n,d}$ and $x \prec y$.)

Even though many new ideas were required, a number of techniques from monotonicity literature turned out to be a good starting point for our investigation of the Lipschitz property. We found this connection between monotonicity and the Lipschitz property surprising because the two properties are defined in terms of different-looking conditions: one is defined on ordered pairs, the other, on unordered pairs; one is about the order relationship, and the other is defined in terms of proximity. We did not discover any reductions between the corresponding testing or reconstruction problems for the two properties. Nonetheless, in one case – for testing functions on the line – we found a formal relationship between the two properties: we show that they are both instances of a class of properties to which the same techniques apply.

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 3.1, 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 $\mathit{image\text{-}diam}(f)$, 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 $O\left(\frac{d \cdot \min\{d, \mathit{image\text{-}diam}(f)\}}{\delta\epsilon}\right)$ time for*[2] *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 3.2) 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[3]. The 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.

Even though the analysis of the tester in the proof of Theorem 1.1 does not apply directly to real-valued functions, by discretizing the values of the functions, we obtain the following corollary for such functions. The corollary is proved in Section 3.1.

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 $O\left(\frac{d \cdot \min\{d, \mathit{image\text{-}diam}(f)\}}{\delta\epsilon}\right)$ 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. *A (possibly 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 $\{0,1,2\}$.*

[2] If $\delta > 1$ then f is Lipschitz iff it is 0-Lipschitz (that is, constant). Testing if a function is constant takes $O(1/\epsilon)$ time.

[3] The main component of our tester – sampling edges uniformly at random and checking for violations of the property – is very natural and was already used in testing monotonicity of functions on the hypercube in [27, 22]. Naturally, the authors of these papers also needed to provide a connection between the number of edges violated by the function and its distance to the property. For the case of functions with Boolean range, they did it by showing that swapping 0 and 1 values on the endpoints of violated edges in one dimension at a time repairs the function. This allowed them to bound the distance of f to a monotone function. The analogous statement for more general ranges (in [22]) is proved by induction on the size of the range. We use the idea of repairing the function one dimension at a time. But many new ideas are needed to make our analysis work. Observe that in the case of the Lipschitz property, functions with Boolean ranges are always Lipschitz, so there is nothing to test. In addition, in this case, not only the size of the range, but also the distances between points in the range play a role. Even though for monotonicity, repairing a function with a range of size greater than 2 in one dimension at a time does not work, this is exactly what we do here.

We prove Theorem 1.3 in Section 3.1.3 using the method presented in [12] of reducing a suitable communication complexity problem to the testing problem. [12] 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 a class of properties of functions on G_n, where G_n is a directed acyclic graph on n vertices. This class includes the Lipschitz property of functions on $\mathcal{L}_n$. (Observe that the Lipschitz property of functions on $\mathcal{L}_n$ and on $\overrightarrow{\mathcal{L}_n}$ are identical.) We extend the monotonicity tester from [11], based on 2-transitive-closure spanners (2-TC-spanners; see Definition 2.4), for functions $f : G_n \to \mathbb{R}$ by abstracting out the requirements on the property needed for the tester and the analysis to work. Our tester works for any property $\mathcal{P}$ of a function $f : G_n \to R$, where R is an arbitrary range, as long as (a) $\mathcal{P}$ can be expressed in terms conditions on pairs of domain points; (b) $\mathcal{P}$ is transitive: namely, for all $x \prec y \prec z$ in the domain[4], whenever (x, y) and (y, z) satisfy the above conditions, so does (x, z); and (c) any function that satisfies the above conditions on a subset of the domain can be extended to a function with the property. We call a property *edge-transitive* if it satisfies (a) and (b), and say it *allows extension* if it satisfies (c). (See Definition 3.5.) Examples of edge-transitive properties include the c-Lipschitz property on directed hypergrids and variants of monotonicity. (See the discussion after Definition 3.5.)

The Lipschitz property for functions on $f : \mathcal{L}_n \to R$ allows extension for most ranges R of interest. We characterize such ranges R (in Claim 3.5) as *discretely metrically convex* metric spaces. Metric convexity is a standard notion in geometric functional analysis (see, e.g., [8]). 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 [8] and its relaxation)**.** *A metric space (R, d_R) is* metrically convex *(respectively,* discretely metrically convex*) if for all points $u, v \in R$ and positive real numbers (respectively, positive integers) α and β satisfying $d_R(u, v) \leq \alpha + \beta$, there exists $w \in R$ such that $d_R(u, w) \leq \alpha$ and $d_R(w, v) \leq \beta$.*

The following theorem, proved in Section 3.2.1, gives an efficient tester for every edge-transitive property that allows extension and, in particular, applies to the Lipschitz property of functions $f : \mathcal{L}_n \to R$.

Theorem 1.4. *Let G_n be a directed graph on n nodes, R be an arbitrary range, and $\mathcal{P}$ be an edge-transitive property of functions $f : G_n \to R$ that allows extension. If G_n has a 2-TC-spanner with $s(n)$ edges, then $\mathcal{P}$ can be tested in time $O(\frac{s(n)}{\epsilon n})$.*

Corollary 1.5. *The Lipschitz property of functions $f : \mathcal{L}_n \to R$ for every discretely metrically convex space R can be tested in time $O\big(\frac{\log n}{\epsilon}\big)$. 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 $G = (V, E)$.*

The following theorem, proved in Section 3.2.2, shows that the upper bound of Corollary 1.5 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, however, that it does not – and should not – apply to the shortest path metric on graphs.)

Theorem 1.6. *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.*

[4] $\prec$ denotes the partial order on the vertices imposed by the edges of G_n. That is, $x \prec y$ for distinct x and y if y is reachable from x in G_n.

The proof of Theorem 1.6 is standard, except for a construction of 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 1.7, which is proved in Section 4.1.

Theorem 1.7 (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.* [3] 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 Section 4.2. The main tool in the analysis is transitive-closure spanners, which were also used in [10] 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 [18]. 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 [18] relies on inspecting the code of the program to verify that it computes a Lipschitz function. Their method might work for some 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. (In order to use a program as an oracle, we need a guarantee that it terminates. This guarantee is also required in [18].) 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. As examples, consider the following three Lipschitz functions: (1) the sum of the values in a Boolean array; (2) the distance of an undirected graph, represented by its Boolean adjacency matrix, to the property of being triangle-free; (3) a function which takes the age of a person and outputs a real vector, where each component is a probability of catching a given disease at that age (presumably, this vector should not change much for people whose ages differ by a year). Our testers apply to all three cases and can be used to approximately certify that programs that claim to compute these functions are indeed computing Lipschitz functions.

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

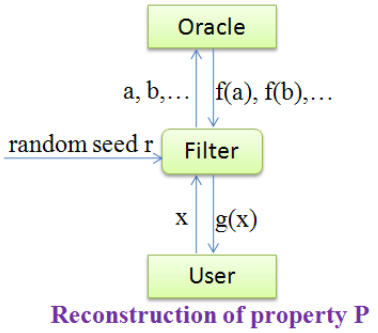


Figure 1: A property reconstructor: g always satisfies the property P

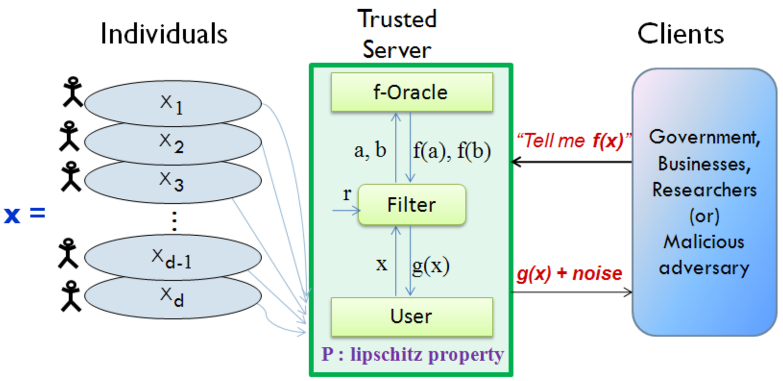


Figure 2: Use of Lipschitz filter in private data analysis

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 [21] pioneering a line of work in cryptography. Subsequently, [23] defined a rigorous notion of privacy, called *differential privacy*, and described the *Laplace mechanism* 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 [34]. 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 [40], allows 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[5]. Crucially, since the filter is local, it guarantees that g does not depend

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

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[6].

In Section 5.2, we instantiate the filter mechanism with our filter from Theorem 1.7 to obtain a private and efficient 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 all elements of the universe U is known. (Note that the total number of people in databases forms a trivial upper bound.) This allows us to evaluate functions defined on multisets privately and efficiently, with the same expected error as in the Laplace mechanism, when the client provides a correct Lipschitz constant. Furthermore, the computation remains 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.

2 Preliminaries

Property Testing. Property testing is concerned with the problem of *approximately* establishing whether certain objects have a desired property or are very "far" from it. We focus on properties of functions over finite domains. Let $\mathcal{U}$ denote the set of all functions on a domain D. A property $\mathcal{P}$ is a subset of $\mathcal{U}$. For example, the Lipschitz property is the set of Lipschitz functions on D. Given a function $f \in \mathcal{U}$, we say f *satisfies* $\mathcal{P}$, if $f \in \mathcal{P}$. Given functions $f, g \in \mathcal{U}$, the *distance* between f and g, denoted $Dist(f, g)$, is the number of points in the domain on which f and g differ. The *relative distance* between f and g is $Dist(f, g)/|D|$. The distance of a function f from a property $\mathcal{P}$, denoted $Dist(f, \mathcal{P})$, is $\min_{g \in \mathcal{P}} Dist(g, f)$. Similarly, the relative distance of f from $\mathcal{P}$, denoted $\epsilon_{\mathcal{P}}(f)$, is $Dist(f, \mathcal{P})/|D|$. We say f is ϵ-far from $\mathcal{P}$ if its relative distance from $\mathcal{P}$ is at least ϵ. 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. In this paper we consider local reconstruction of the Lipschitz property. The model of local reconstruction was defined in [43], and the variant we consider was given in [10].

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 $A_{f,\rho}(x) \in R$. (Note that a local filter has no internal state to store previously made queries.) The function $g(x) = A_{f,\rho}(x)$*

[6] The reason we do not insist that the filter be distance-preserving is because we call it only on one database x. If it were distance-preserving, 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, tt is not hard to see that our filter never changes $f(x)$ by more than $\max_y \{|f(y) - f(x)| + d_G(x, y)\}$.

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.*

In answering query $x \in D$, the filter A may ask for values of f at domain points of its choice (possibly adaptively) using its oracle access to f. Each such access made to the oracle is called a *lookup* to distinguish it from the client query x. A local filter is *nonadaptive* if the set of domain points that the filter looks up to answer an input query x does not depend on answers given by the oracle.

In [43], the authors also require that g is sufficiently close to f: with high probability (over the choice of ρ), $Dist(g, f) \leq B(n) \cdot Dist(f, \mathcal{P})$, where $B(n)$ is called the *error blow up*. Our definition does not have this requirement because in the application to data privacy, it is not necessary: the error blow up corresponds to the penalty incurred by a client misreporting a Lipschitz constant of the submitted function.

Facts about Lipschitz functions. If f is not Lipschitz, then for some pair $(x, y) \in D \times D$, the Lipschitz condition is violated, namely, $d_R(f(x), f(y)) > d_D(x, y)$. Such a pair is called *violated*.

Definition 2.2. *A function $f : D \to R$ is Lipschitz on $D' \subseteq D$ if there are no violated pairs in $D' \times D'$.*

We note the following standard fact about extending partial Lipschitz functions.

Fact 2.1 (Lemma 1.1, [8])**.** *Consider a function $f : D \to \mathbb{R}^k$ between metric spaces (D, d_D) and $(\mathbb{R}^k, \ell_\infty)$. If f is* Lipschitz on *$D' \subseteq D$, one can make f Lipschitz (on the entire domain) by modifying it only on $D \setminus D'$.*

In this work, we focus on functions over discrete domains which can be represented by a (usually undirected) graph G equipped with the shortest-path metric $d_G(\cdot, \cdot)$. We say that an edge (x, y) in G is *violated* if (x, y) is a violated pair. Observe that a function $f : G \to R$, that maps vertices of G to R, is Lipschitz iff $d_R(f(x), f(y)) \leq d_G(x, y)$ for all edges (x, y) in G. Given this observation, it is easy to see that a function $f : G \to R$ defined on the vertices of a *directed* graph G is Lipschitz iff it is Lipschitz with respect to the underlying undirected graph. (However, this artificial introduction of directions gives properties which, in general, do not allow extension; see Definition 3.5 and discussion following it.)

When we talk about properties defined on (acyclic) directed graphs, we identify their vertices with elements of the corresponding partial order.

Definition 2.3 (Comparable and incomparable elements)**.** *Let G be a partially ordered set equipped with a partial order $\prec$. Elements $a, b \in G$ are* comparable *if $a \preceq b$ or $b \preceq a$. Otherwise, a and b are incomparable.*

Transitive-Closure Spanners. Transitive-closure spanners (see [37] for a survey on the topic) are used in Sections 3.2.1 and 4.2.

Definition 2.4 (k-TC-spanner, [11])**.** *Given a directed graph $G = (V, E)$ and an integer $k \geq 1$, a k-*transitive-closure-spanner (k-TC-spanner) *of G is a directed graph $H = (V, E_H)$ such that:* (a) *E_H is a subset of the edges in the transitive closure of G;* (b) *for all vertices $x, y \in V$, if $d_G(x, y) < \infty$, then $d_H(x, y) \leq k$.*

Directed hypergrids were defined in Section 1.1. The following bounds from [9, 10] on the size of 2-TC-spanners of these graphs are used in Section 4.2 to prove lower bounds for local Lipschitz filters.

Lemma 2.2. *A 2-TC-spanner of $\overrightarrow{\mathcal{H}}_{n,d}$ has $\Omega\left(\frac{n^d (\ln n - 1)^d}{(4\pi)^d}\right)$ edges* [9]*. A 2-TC-spanner of $\overrightarrow{\mathcal{H}}_d$ has $\Omega(2^{cd})$ edges, where $c \approx 1.1620$* [10].

3 Property Testers

3.1 Hypercube: Testing if a function on $\mathcal{H}_d$ is Lipschitz

In this section, first we show how to test if a function $f : \mathcal{H}_d \to \delta\mathbb{Z}$ is Lipschitz and prove Theorem 1.1. Then we derive Corollary 1.2 on (a relaxation of) testing if a function $f : \mathcal{H}_d \to \mathbb{R}$ is Lipschitz. At the end (in Section 3.1.3), we prove Theorem 1.3 which gives a lower bound on the query complexity of a Lipschitz tester on the hypercube.

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.

Recall that a function is Lipschitz if its values on the endpoints of every edge differ by at most 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* $\mathit{image\text{-}diam}(f)$*, 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 tester runs an algorithm that approximates the image diameter of the input function in the sense of the following lemma, and rejects if it is greater than d. The algorithm computes the image diameter of the input function f, restricted to random samples.

Lemma 3.1. *There is an algorithm that, given a function* $f : D \to \mathbb{R}$ *and* $\epsilon \in (0, 1]$*, outputs* $r \in \mathbb{R}$ *such that* $r \le \mathit{image\text{-}diam}(f)$ *and with probability* $\ge \frac{5}{6}$ *the function* f *is* ϵ*-close to having image diameter at most* r*. Moreover, the algorithm runs in* $O(1/\epsilon)$ *time.*

Lemma 3.1 is proved in Section 3.1.2. After rejecting functions on which the algorithm of Lemma 3.1 returns a value greater than d, our tester samples hypercube edges uniformly at random to check if any of them are violated. The main tool in the analysis of our test is the following lemma, proved in Section 3.1.1.

Lemma 3.2 (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* $V(f) \ge \delta\epsilon \cdot 2^{d-1} / \mathit{image\text{-}diam}(f)$.

We now present our proof of Theorem 1.1, on testing if a function $f : \{0,1\}^d \to \delta\mathbb{Z}$ is Lipschitz.

Proof of Theorem 1.1. Let SAMPLE-DIAMETER be the algorithm given by Lemma 3.1. The following tester of the Lipschitz property of functions on $\mathcal{H}_d$ is used to prove the theorem:

```
LIPSCHITZ-TEST-HYPERCUBE($f : \{0,1\}^d \to \delta\mathbb{Z}, d, \delta, \epsilon$)
1  Let $r \leftarrow$ SAMPLE-DIAMETER($f, \epsilon/2$). If $r > d$, reject.
2  Select $s = \lceil (2 \cdot dr)/\delta\epsilon \rceil$ edges uniformly and independently at random from the hypercube $\mathcal{H}_d$.
3  If any of the selected edges $\{x, y\}$ are violated, i.e., $|f(x) - f(y)| > 1$, reject; otherwise, accept.
```

HYPERCUBE-LIPSCHITZ-TEST always accepts a Lipschitz function. Consider a function f which is ϵ-far from the Lipschitz property. Let E be the event that the output r of the procedure SAMPLE-DIAMETER is such that the function f is $\epsilon/2$-far from having image diameter r. By Lemma 3.1, $\Pr[E] \le 1/6$. If $r > d$ then the tester correctly rejects on line 1 because, by Lemma 3.1, $r \le \mathit{image\text{-}diam}(f)$, and a Lipschitz function on $\mathcal{H}_d$ must have image diameter at most d.

It remains to consider the case when $r \le d$. Conditioned on E not happening (denoted $\overline{E}$), there is a function h with $\mathit{image\text{-}diam}(h) \le r$ such that $\mathit{dist}(f, h) < \epsilon/2$, where $\mathit{dist}(f, h)$ denotes the fraction of points on which f and h differ. In the following, assume E does not occur. Let $a_{min} = \min_{x \in \{0,1\}^d} h(x)$

and $a_{max} = \max_{x \in \{0,1\}^d} h(x)$. Consider a function g, obtained from f by setting $g(x) = a_{min}$ when $f(x) < a_{min}$, $g(x) = a_{max}$ when $f(x) > a_{max}$, and $g(x) = f(x)$ for remaining $x \in \{0,1\}^d$. Then $\textit{image-diam}(g) \leq r$ and $dist(f, g) < \epsilon/2$. Observe that an edge can be violated by g only if it is violated by f. That is, $V(f) \geq V(g)$. (Recall that $V(f)$ denotes the number of edges violated by f.) By the triangle inequality, the relative distance from g to the Lipschitz property is $\epsilon_{lip}(g) \geq \epsilon_{lip}(f) - \epsilon/2 \geq \epsilon/2$. Since g is $\epsilon/2$-far from Lipschitz and $\textit{image-diam}(g) \leq r$, Lemma 3.2 asserts that $V(g) \geq \delta\epsilon \cdot \frac{2^{d-1}}{r}$. Therefore, the fraction of edges in $\mathcal{H}_d$ violated by f is $V(f)/|E(\mathcal{H}_d)| \geq V(g)/(2^{d-1}d) \geq \delta\epsilon/(dr)$. Let $Reject$ denote the event that HYPERCUBE-LIPSCHITZ-TEST rejects f. Since HYPERCUBE-LIPSCHITZ-TEST samples $2dr/\delta\epsilon$ edges in $\mathcal{H}_d$ uniformly and independently, $\Pr[Reject|\overline{E}] \geq 4/5$. Therefore, $\Pr[Reject] \geq \Pr[Reject|\overline{E}] \cdot \Pr[\overline{E}] \geq (4/5) \cdot (5/6) = 2/3$, as required.

Finally, observe that the running time is $O(1/\epsilon)$ if SAMPLE-DIAMETER returns $r > d$. Otherwise, it is $O(dr/(\delta\epsilon))$, where $r \leq \min\{d, \textit{image-diam}(f)\}$. This completes the proof of the theorem. □

Next we prove the corollary on (a relaxation of) testing if a function $f : \mathcal{H}_d \to \mathbb{R}$ is Lipschitz.

Proof of Corollary 1.2. Let $\delta' = \delta/2$ and $f' : \mathcal{H}_d \to \delta'\mathbb{Z}$ be the function defined by $f'(x) = \lfloor f(x) \rfloor_{\delta'}$ for all $x \in \{0,1\}^d$. (The operator $\lfloor\ \rfloor_\delta$ is defined in the beginning of Section 3.1.1.) Then $f(x) - \delta' \leq f'(x) \leq f(x)$ for all $x \in \{0,1\}^d$. If f is Lipschitz then f' is $(1+\delta')$-Lipschitz because $|f'(x) - f'(y)| \leq |f(x) - f(y)| + \delta' \leq 1 + \delta'$ for each edge $\{x, y\}$ of the hypercube $\mathcal{H}_d$. Next we show that when f is ϵ-far from $(1 + 2\delta')$-Lipschitz then f' is ϵ-far from $(1+\delta')$-Lipschitz. Suppose to the contrary that f' is $(1+\delta')$-Lipschitz on a set $S \subseteq \{0,1\}^d$ of size greater than $(1-\epsilon)2^d$. (See Definition 2.2 and Fact 2.1.) Since $f'(x) \leq f(x) \leq f'(x) + \delta'$, function f is $(1+2\delta')$-Lipschitz on S, a contradiction.

Thus, we can use the Lipschitz testers of Theorem 1.1 with inputs $f'/(1+\delta'), d, \delta'/(1+\delta')$ and ϵ to distinguish between Lipschitz f and f that is ϵ-far from $(1+\delta)$-Lipschitz, proving the corollary. □

3.1.1 Averaging Operator A_i

This section is devoted to Lemma 3.2, the main tool in the analysis of the tester of the Lipschitz property on the hypercube. To prove Lemma 3.2, 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 $\lfloor x \rfloor_\delta$ (respectively, $\lceil x \rceil_\delta$) be the smallest (respectively, largest) value in $\delta\mathbb{Z}$ not greater (respectively, 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*

$$A_i[f](x) = \left\lfloor \frac{f(x) + f(y)}{2} \right\rfloor_\delta \text{ and } A_i[f](y) = \left\lceil \frac{f(x) + f(y)}{2} \right\rceil_\delta .$$

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.*

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

$$\delta V^i(f) \leq VS^i(f) \leq V^i(f) \cdot \textit{image-diam}(f). \tag{1}$$

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.

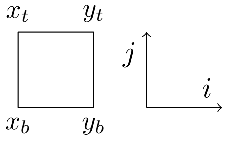
Lemma 3.3. *For all $i, j \in [d]$, where $i \neq j$, and every function $f : \{0,1\}^d \to \delta\mathbb{Z}$, applying the* averaging *operator A_i does not increase the violation score in dimension j, i.e., $VS_j(A_i[f]) \leq VS_j(f)$.*

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 3.3 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 3.3 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. In this proof, we use the fact that $1/\delta$ is integral, discussed in the beginning of Section 3.1.



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, $f(y_t) \geq f(x_t) + 1 + \delta$. 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 $f(x_b) \geq f(y_b) - 1$ 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 $f(x_t) \geq f(x_b) + 1$. Combining the three inequalities derived so far, we get $f(y_t) \geq f(x_t) + 1 + \delta \geq f(x_b) + 2 + \delta \geq f(y_b) + 1 + \delta$. 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. □

Proof of Lemma 3.2. 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 $\frac{2}{\delta} \cdot V(f) \cdot \textit{image-diam}(f)$ 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 of the averaging operator A_i, 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 3.3, 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, $Dist(f, f_d)$. For all $i \in [d]$,

$$\begin{aligned} Dist(f_{i-1}, f_i) &= Dist(f_{i-1}, A_i[f_{i-1}]) \leq 2 \cdot V^i(f_{i-1}) \leq \frac{2}{\delta} \cdot VS^i(f_{i-1}) \leq \frac{2}{\delta} \cdot VS^i(f) \\ &\leq \frac{2}{\delta} \cdot V^i(f) \cdot \mathit{image\text{-}diam}(f). \end{aligned} \tag{2}$$

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 3.3, operators A_j for $j \neq i$ do not increase the violation score in dimension i. The distance from f to f_d is

$$Dist(f, f_d) \leq \sum_{i \in [d]} Dist(f_{i-1}, f_i) \leq \sum_{i \in [d]} \frac{2}{\delta} \cdot V^i(f) \cdot \mathit{image\text{-}diam}(f) = \frac{2}{\delta} \cdot V(f) \cdot \mathit{image\text{-}diam}(f). \tag{3}$$

The two inequalities above follow from the triangle inequality and (3), respectively.

Consider a function f which is ϵ-far from the Lipschitz property. Since f_d is Lipschitz, $Dist(f, f_d) \geq \epsilon \cdot 2^d$. Together with (3), it gives $V(f) \geq \epsilon\delta \cdot 2^{d-1} / \mathit{image\text{-}diam}(f)$, as required. □

3.1.2 Analyzing the image diameter of a sample

Next we present SAMPLE-DIAMETER and its analysis, thus proving Lemma 3.1.

```
SAMPLE-DIAMETER($f : D \to \mathbb{R}, \epsilon$)
Let $s = \lceil 6/\epsilon \rceil$. Select samples $\mathbf{z} = z_1, \ldots, z_s$ from $D$ uniformly and independently at random.
Return $r = \max_{i=1}^{s} f(z_i) - \min_{i=1}^{s} f(z_i)$.
```

Proof of Lemma 3.1. SAMPLE-DIAMETER always returns r no larger than $\mathit{image\text{-}diam}(f)$. To show that with probability $\geq \frac{5}{6}$, output r is such that the function f is ϵ-close to having image diameter at most r, sort the points in the domain D of the input function f in non-decreasing order according to their f-values. Let L be the first (respectively, let R be the last) $\lceil \epsilon \cdot |D|/2 \rceil$ points in the sorted list. Define $x_1 = \operatorname{argmax}_{x \in L} f(x)$ and $x_2 = \operatorname{argmin}_{x \in R} f(x)$. Let E_1 (respectively, E_2) denote the event that the sampled sequence $\mathbf{z}$ contains no element of L (respectively, R). Observe that if $\mathbf{z}$ contains an element of L and an element of R, that is, $\overline{E_1 \cup E_2}$ holds, then f is ϵ-close to having diameter at most r. This is because by redefining f only on points in $(L \cup R) \setminus \{x_1, x_2\}$ whose f-values are smaller or larger than f-values on all the samples, we get a function with image diameter at most r. The probability that it fails to happen is

$$\Pr(E_1 \cup E_2) \leq 2 \cdot \Pr(E_1) \leq 2 \cdot \left(1 - \frac{\epsilon}{2}\right)^{\lceil \frac{6}{\epsilon} \rceil} \leq 2 \cdot \left(e^{-\frac{\epsilon}{2}}\right)^{\frac{6}{\epsilon}} \leq \frac{1}{6}.$$

The first inequality above uses the union bound and symmetry. The third inequality holds since $|L|/|D| \geq \frac{\epsilon}{2}$. The rest is standard. □

3.1.3 Lower bound on the Lipschitz tester for the hypercube

In this section, we prove Theorem 1.3 which gives a lower bound on the query complexity of an (adaptive, two-sided error) Lipschitz tester for the hypercube. The proof uses the method presented in [12] of reducing a suitable communication complexity problem to the testing problem.

Proof of Theorem 1.3. Consider the following communication game between Alice and Bob in the public randomness model, where the players generate messages based on random bits they both see. Alice has an input $A \subseteq [d]$, Bob has an input $B \subseteq [d]$, and they would like to compute the set-disjointness function $DISJ_d(A, B)$, which is 1 if $A \cap B = \emptyset$ and 0 otherwise. It is well-known [32, 6, 38] that $R(DISJ_d)$, the minimum number of bits Alice and Bob must communicate for them both to compute $DISJ_d(A, B)$ with probability at least $2/3$ on any input pair (A, B), is $\Omega(d)$.

Alice and Bob can reduce the problem of computing $DISJ_d$ to testing the Lipschitz property as follows. Note that a set $S \subseteq [d]$ is uniquely determined by the parity function $\chi_S : \{0,1\}^d \to \{-1,1\}$ given by $\chi_S(x) = -1^{\sum_{i \in S} x_i}$. Alice forms the function $f = \chi_A$ and Bob forms the function $g = \chi_B$. Claim 3.4 shows that the *joint* function $h = (f + g)/2$ is Lipschitz if the sets do not intersect, and $1/4$-far from Lipschitz otherwise. Thus, the players can determine if their sets intersect by both running the same tester for the Lipschitz property on h. Whenever Alice's tester queries $h(x)$, she sends $f(x)$ to Bob, and whenever Bob's tester queries $h(x)$, he sends $g(x)$ to Alice. Both use the received message to compute $h(x)$.

Let $q(d)$ be the query complexity of testing the Lipschitz property of functions of the form $h : \{0,1\}^d \to \{-1, 0, 1\}$. If the players run an optimal tester, they exchange $2q(d)$ messages, 1 bit each. That is, $R(DISJ_d) \leq 2q(d)$. The claimed bound then follows from the $\Omega(d)$ bound on $R(DISJ_d)$. □

The following claim was used in the proof of Theorem 1.3.

Claim 3.4. *Given subsets $A, B \subseteq [d]$, let $h : \{0,1\}^d \to \{-1, 0, 1\}$ be the function defined by $h(x) = (\chi_A(x) + \chi_B(x))/2$. Then h is Lipschitz if $A \cap B = \emptyset$, and 1/4-far from Lipschitz otherwise.*

Proof. Consider an arbitrary dimension $j \in [d]$ and fix an arbitrary edge $\{x, y\}$ along dimension j such that $x_j = 0$ and $y_j = 1$. One may verify that for any subset $S \subseteq [d]$, $\chi_S(x) - \chi_S(y) = 2 \cdot \chi_S(x) \cdot |S \cap \{j\}\,|$. This implies that $|h(x) - h(y)| \leq |A \cap \{j\}\,| + |B \cap \{j\}\,|$. Therefore, if A and B are disjoint, $|h(x) - h(y)| \leq 1$ for each edge $\{x, y\}$ of the hypercube (thus implying h is Lipschitz). Now suppose A and B intersect and consider some $j \in A \cap B$. Now, for any edge $\{x, y\}$ as above, $|h(x) - h(y)| = |\chi_A(x) + \chi_B(x)|$. This is equal to 2 whenever $\chi_A(x) = \chi_B(x)$, which holds for at least half of the vertices $x \in \{0,1\}^d$ with $x_j = 0$. Moreover, the corresponding violated edges $\{x, y\}$ form a matching. Therefore, in this case the function h is 1/4-far from Lipschitz. □

3.2 Line Graph: Testing if a function on $\mathcal{L}_n$ is Lipschitz

3.2.1 A tester for edge-transitive properties that allow extension

In this section, we prove Theorem 1.4 and Corollary 1.5. We start by giving the definition and examples of edge-transitive properties that allow extension. Recall the definition of comparable and incomparable elements (Definition 2.3).

Definition 3.5. *Let G be a directed graph and R be an arbitrary range. A property $\mathcal{P}$ of functions $f : G \to R$ is* edge-transitive *if the following conditions hold:*

1. *It can be expressed in terms of requirements on pairs of comparable domain points, i.e., $f \in \mathcal{P}$ iff f satisfies given requirements on $f(x), f(y)$ for all comparable vertices x, y in G. A pair (x, y) is called* violated *(by f) if the corresponding requirement on $f(x), f(y)$ is not satisfied.*
2. *For all vertices $x \prec y \prec z$ in G, whenever (x, y) and (y, z) are not violated, neither is (x, z).*

An edge-transitive property $\mathcal{P}$ allows extension *if every partial function, which is defined on a subset D' of the domain and violates no pairs in $D' \times D'$, can be extended to a function $f \in \mathcal{P}$ over the entire domain.*

Examples of edge-transitive properties include the c-Lipschitz property of functions over hypergrids and variants of monotonicity. Recall that the c-Lipschitz property was defined in terms of pairs of domain elements. Specifically, a pair (x, y) is *violated* if $d_R(f(x), f(y)) > c \cdot d_G(x, y)$. If (x, y) and (y, z) are not violated then, by the triangle inequality, $d_R(f(x), f(z)) \leq d_R(f(x), f(y)) + d_R(f(y), f(z)) \leq c \cdot d_G(x, y) + c \cdot d_G(y, z) = c \cdot d_G(x, z)$ and, consequently, (x, z) is not violated. (The last equality uses the fact that G is a hypergrid.) Therefore, the c-Lipschitz property is edge-transitive. Another example of an edge-transitive property is monotonicity, which is defined by $f \in \mathcal{P}$ if $f(x) \leq f(y)$ for all $x \prec y$. Edge-transitive variants on monotonicity include strict monotonicity, where the requirements are $f(x) < f(y)$ for all $x \prec y$, and c-monotonicity of functions over hypergrids, where the requirements are $f(x) \leq c^{d_G(x,y)} \cdot f(y)$ for all $x \prec y$; G being the hypergrid graph.

While extendability is rarely an issue for variants of monotonicity, it is the reason that the tester in this section is not applicable for the Lipschitz property of functions on other domains of interest, such as the hypercube $\overrightarrow{\mathcal{H}}_d$ and the hypergrid $\overrightarrow{\mathcal{H}}_{n,d}$. Monotonicity, strict monotonicity and c-monotonicity of functions over hypergrids allow extension when the range of functions is $\mathbb{R}$. When the range is $\mathbb{Z}$, monotonicity and c-monotonicity of functions over hypergrids still allow extension, but strict monotonicity, the way we defined it above, does not. However, it allows extension if the requirements are replaced by $f(x) \leq f(y) + d_G(x, y)$ for all $x \prec y$. Therefore, the tester of this section applies to all these monotonicity variants with the ranges we mentioned. For the Lipschitz property, the situation is fundamentally different. If the (directed) domain graph contains two incomparable vertices x and y, which are connected in the underlying undirected graph, then the Lipschitz property of functions over this domain does not allow extension. To see this, denote the distance from x to y in the underlying undirected graph by d. If $f(x) = 0$ and $f(y) = d + 1$, there is no way to assign values of f on all other vertices to ensure that it is Lipschitz, even though no requirement on comparable vertices is violated. Note that while the Lipschitz property of functions on *undirected* graphs allows extension, say, for range $\mathbb{R}$, it is not edge-transitive in the sense of Definition 3.5. Fortunately, when the domain of functions is $\overrightarrow{\mathcal{L}_n}$, for many ranges of interest, such as $\mathbb{R}^k$, equipped with ℓ_1, ℓ_2 or ℓ_∞, $\mathbb{Z}^k$, equipped with ℓ_1 or ℓ_∞, and the shortest path metric d_G on all unweighted graphs G, the Lipschitz property allows extension. We characterize ranges R for which the Lipschitz property of functions $f : [n] \to R$ allows extension in Claims 3.5.

Next we prove Theorem 1.4 that gives a tester for every edge-transitive property of functions $f : G_n \to R$ that allows extension, where G_n is a directed acyclic graph and R is an arbitrary range. It extends the monotonicity tester from [11] for functions $f : G_n \to \mathbb{R}$, based on 2-TC-spanners (see Definition 2.4).

Proof of Theorem 1.4. Let $H = (V, E)$ be a 2-TC-spanner of G_n with $s(n)$ edges. The following tester works for all edge-transitive properties $\mathcal{P}$ that allow extension. It selects $\lceil 4s(n)/(\epsilon n) \rceil$ edges uniformly and independently from H and queries f on their endpoints. The tester rejects if some selected edge (x, y) is violated by f with respect to $\mathcal{P}$, and accepts otherwise.

This tester always accepts a function $f \in \mathcal{P}$. Consider the case when f is ϵ-far from $\mathcal{P}$. Let $V_1 \subseteq V$ be the set of endpoints of edges in H violated by f, and $V_2 = V \setminus V_1$. We claim that no pairs $(x, y) \in V_2 \times V_2$ are violated. To see this, consider such a pair with $x \prec y$. Since H is a 2-TC-Spanner of G_n, it contains edges (x, z) and (z, y), where $x \preceq z \preceq y$. Edges (x, z) and (z, y) are not violated because $x, y \in V_2$. Since f is edge-transitive, (x, y) is also not violated. Therefore, f violates no pairs in $V_2 \times V_2$. Since $\mathcal{P}$ allows extension and f is ϵ-far from $\mathcal{P}$, it implies that $|V_1| \geq \epsilon n$. Since each violated edge in H contributes at most 2 distinct endpoints to V_1, the number of violated edges is at least $\epsilon n/2$. Consequently, a uniformly selected edge in H is violated with probability at least $\epsilon n/(2s(n))$ because H has at most $s(n)$ edges. Since the test samples $\lceil 4s(n)/(\epsilon n) \rceil$ edges uniformly and independently, it finds a violated edge and, therefore, rejects with probability at least 2/3, as required. □

Next we prove Corollary 1.5 that gives a tester for the Lipschitz property of functions $f : \mathcal{L}_n \to R$ for every discretely metrically convex space R.

Proof of Corollary 1.5. A sparse 2-TC-spanner of the directed line $\overrightarrow{\mathcal{L}}_n$ with at most $n \log n$ edges can be constructed greedily. This construction appeared implicitly or explicitly as a special case of more general constructions in [45, 17, 16, 4, 20, 13, 44, 22, 5]. It is surveyed as a stand-alone construction in [37]. Since the Lipschitz property is edge-transitive and, by Claim 3.5, allows extension whenever R is discretely metrically convex, Theorem 1.4 implies the first part of the corollary. Then the second part with examples of spaces R follow from Claim 3.6. □

Next we prove the claims used in the proof of Corollary 1.5. Claim 3.5 characterizes ranges R for which the Lipschitz property of functions $f : [n] \to R$ allows extension. Claim 3.6 gives examples of such ranges.

Claim 3.5. *Metric space (R, d_R) is* discretely metrically convex *if and only if for every $D' \subseteq [n]$, every Lipschitz function $f : D' \to R$ can be extended to a Lipschitz function on the entire domain $[n]$.*

Proof. First assume that (R, d_R) is discretely metrically convex. Fix an arbitrary $x \in [n] \setminus D'$. We show how to extend f to x such that the extension is still Lipschitz. Let ℓ (respectively, r) be the D'-vertex closest to x on the left (respectively, right). If either ℓ or r does not exist, set $f(x)$ to the value of f at the other vertex. Otherwise, set $f(x)$ to some $w \in R$ satisfying $d_R(f(\ell), w) \leq x - \ell$ and $d_R(w, f(r)) \leq r - x$. Such a point w exists because R is discretely metrically convex and $d_R(f(\ell), f(r)) \leq r - \ell = (r - x) + (x - \ell)$.

It remains to prove that the extended f is Lipschitz. For that, it is sufficient to show that it is Lipschitz on $S = \{\ell, x, r\}$ (that is, there are no violated pairs in $S \times S$; see Definition 2.2) because for every $y \in D'$, one of ℓ or r lies on the shortest path between x and y. If one of ℓ or r does not exist, f is trivially Lipschitz on S. Otherwise, the definition of $f(x)$ guarantees that $d_R(f(x), f(\ell)) \leq x - \ell$ and $d_R(f(x), f(\ell)) \leq r - x$, implying that f is Lipschitz on S.

For the other direction, assume that every R-valued partial function on $[n]$ which is Lipschitz (with respect to d_R) can be extended to a Lipschitz function on the entire domain. We show R is discretely metrically convex. Fix $u, v \in R$ satisfying $d_R(u, v) \leq \alpha + \beta$ for positive integers α and β. Now, consider a partial function $f : [\alpha+\beta+1] \to R$ such that $f(1) = u$ and $f(\alpha+\beta+1) = v$. Since $d_R(u, v) \leq \alpha+\beta$, f is Lipschitz on the set $\{1, \alpha + \beta + 1\}$. By our assumption, the partial function can be extended to a Lipschitz function $\tilde{f}$ on the entire domain. Now, $w = \tilde{f}(\alpha) \in R$ satisfies $d_R(u, w) \leq \alpha$ and $d_R(w, v) \leq \beta$ because $\tilde{f}$ is Lipschitz. □

Claim 3.6 (Examples of discretely metrically convex metric spaces.)**.** *The following metric spaces are discretely metrically convex: $(\mathbb{R}^k, \ell_p)$ 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 (unweighted) graphs $G = (V, E)$.*

Proof. For a point $\mathbf{u} \in \mathbb{R}^k$ and $p \in [1, \infty)$, let $||\mathbf{u}||_p = \left(\sum_{i\in[k]} |u_i|^p\right)^{1/p}$ denote the *ℓ_p-norm* of $\mathbf{u}$. The ℓ_p metric on $\mathbb{R}^k$ is defined by $d_{\ell_p}(\mathbf{u}, \mathbf{v}) = ||\mathbf{u} - \mathbf{v}||_p$. Given elements $\mathbf{u}, \mathbf{v} \in \mathbb{R}^k$ and positive real numbers α and β satisfying $||\mathbf{u} - \mathbf{v}||_p \leq \alpha + \beta$, let $\mathbf{w} = \frac{\alpha\mathbf{v}+\beta\mathbf{u}}{\alpha+\beta}$. Then $d_R(u, w) \leq \alpha$ and $d_R(w, v) \leq \beta$. This shows that $(\mathbb{R}^k, \ell_p)$ for $p \in [1, \infty)$ is metrically convex. Fact 2.1 and Claim 3.5 imply metric convexity of $(\mathbb{R}^k, \ell_\infty)$. Recall that metric convexity implies discrete metric convexity.

We also observe that the shortest path metric d_G on an (unweighted) graph $G = (V, E)$ is discretely metrically convex. Specifically, suppose $u, v \in V$ satisfy $d_G(u, v) \leq \alpha + \beta$ for positive integers α and β. If $\alpha \geq d_G(u, v)$, then trivially $w = v$ satisfies $d_G(u, w) \leq \alpha$ and $d_G(w, v) \leq \beta$. Otherwise, let w be the vertex at distance α from u on a shortest path between u and v. Such a vertex exists because $\alpha < d_G(u, v)$. Then $d_G(u, w) + d_G(w, v) = d_G(u, v) \leq \alpha + \beta$ implies $d_G(w, v) \leq \beta$. Thus, w is the required vertex. In particular, $\mathbb{Z}^k$ equipped with ℓ_1 metric (which can be viewed as a k-dimensional hypergrid) is *discretely* metrically convex.

Finally, $(\mathbb{Z}^k, \ell_\infty)$ is discretely metrically convex because in each coordinate, $(\mathbb{Z}, \ell_\infty)$ is discretely metrically convex. This holds because ℓ_1 metric and ℓ_∞ metric are identical on $\mathbb{Z}$. Specifically, suppose $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^k$ and $\|\mathbf{u} - \mathbf{v}\|_\infty \le \alpha + \beta$ for positive integers α and β. Then, by definition of ℓ_∞, we have $\max_{j \in [k]} |\mathbf{u}_j - \mathbf{v}_j| \le \alpha + \beta$. Therefore, $|\mathbf{u}_j - \mathbf{v}_j| \le \alpha + \beta$ for each $j \in [k]$. Since $(\mathbb{Z}, \ell_1)$ is discretely metrically convex, for each $j \in [k]$ there exists $w_j \in \mathbb{Z}$ such that $|\mathbf{u}_j - w_j| \le \alpha$ and $|w_j - \mathbf{v}_j| \le \beta$. Define $\mathbf{w}$ by $\mathbf{w}_j = w_j$. Then $\|\mathbf{u} - \mathbf{w}\|_\infty = \max_{j \in [k]} |\mathbf{u}_j - \mathbf{w}_j| \le \max_{j \in [k]} \alpha \le \alpha$. Similarly, $\|\mathbf{w} - \mathbf{v}\|_\infty \le \beta$, thus proving that $(\mathbb{Z}^k, \ell_\infty)$ is discretely metrically convex. □

3.2.2 Lower bound on nonadaptive one-sided error Lipschitz tester for the line

In this section, we prove Theorem 1.6.

Proof of Theorem 1.6. Recall that a function is Lipschitz on a set $D' \subseteq [n]$ if it violates no pairs in $D' \times D'$. (See Definition 2.2.) Observe that a 1-sided error tester must accept if the input function is Lipschitz on the query points because, by Fact 2.1, every such function can be extended to a Lipschitz function. To prove our lower bound, we use Yao's principle. Namely, we define a distribution $\mathcal{N}$ on input functions $f : [n] \to \mathbb{N}$ which are 1/4-far from Lipschitz, and show that for every fixed query set $Q \subset [n]$ of size $o(\log n)$, a function f drawn from $\mathcal{N}$ is Lipschitz on Q with probability more than 1/3.

$$\boxed{0}\boxed{1}\boxed{1}\boxed{1}\;\boxed{2}\boxed{1}\boxed{1}\boxed{0}\;\boxed{0}\boxed{1}\boxed{1}\boxed{1}\;\boxed{2}\boxed{1}\boxed{1}\boxed{0}\;\boxed{0}\boxed{1}\boxed{1}\boxed{1}\;\boxed{2}\boxed{1}\boxed{1}\boxed{0}\;\boxed{0}\boxed{1}\boxed{1}\boxed{1}\;\boxed{2}\boxed{1}\boxed{1}\boxed{0}$$

Figure 3: The discrete derivative function, Δf_i.

Assume $n = 2^\ell$. Distribution $\mathcal{N}$ is uniform over functions f_i indexed by $i \in [\ell]$. Fix an $i \in [\ell]$. Let $\mathcal{P}_i$ denote the partition of $[n]$ into intervals of size 2^i. Namely, $\mathcal{P}_i$ consists of intervals $I^i_j = [1+(j-1)2^i, j2^i]$ for all $j \in [n/2^i]$. We describe the input function f_i by defining its *discrete derivative* function $\Delta f_i : [n] \to \mathbb{N}$. That is, $f_i(x) = \sum_{y \in [x]} \Delta f_i(y)$ for all $x \in [n]$. We define Δf_i as follows: for all intervals $I^i_j = [x, y] \in \mathcal{P}_i$ with odd j (respectively, even j), set $\Delta f_i(x) = 0$ and $\Delta f_i(y) = 1$ (respectively, $\Delta f_i(x) = 2$ and $\Delta f_i(y) = 0$). For remaining $z \in [n]$, set $\Delta f_i(z) = 1$. (See Figure 3.)

Now we show that the resulting function f_i is 1/4-far from Lipschitz. Let $L^i_j = (1 + (j-1)2^i, j2^i]$ and $R^i_j = [1 + (j-1)2^i, j2^i)$ denote the half-open intervals corresponding to I^i_j. Observe that for each odd j, all pairs $(x, y) \in L^i_j \times R^i_{j+1}$ are violated by f_i. Therefore, to make f_i Lipschitz, we have to redefine it on all points in L^i_j or on all points in R^i_{j+1}, that is, on at least 1/4 of points in $I^i_j \cup I^i_{j+1}$. Thus, f_i is 1/4-far from Lipschitz.

Finally, observe that for all i and all $x, y \in [n]$ such that $x < y$, the pair (x, y) is violated by f_i iff $x \in L^i_j$ and $y \in R^i_{j+1}$ for some odd j. That is, each such pair (x, y) is violated by at most one function f_i.

Let $a_1 < \cdots < a_q$ be the queries in some fixed set $Q \subset [n]$. A function f is Lipschitz on Q iff (a_t, a_{t+1}) is not violated for all $t \in [q-1]$. When f is chosen from $\mathcal{N}$, $\Pr[f \text{ violates } (a_t, a_{t+1})] \le 1/\ell$ for each $t \in [q-1]$. By the union bound, $\Pr[f \text{ is not Lipschitz on } Q] \le \sum_{t \in [q-1]} \Pr[f \text{ violates } (a_t, a_{t+1})] \le (q-1)/\ell$. When $q \le 2(\log n)/3$, this is less than 2/3. That is, every deterministic 1-sided error nonadaptive test with q queries fails with probability greater than 1/3 on an input function drawn from $\mathcal{N}$. □

4 Local Reconstruction of the Lipschitz property

4.1 Constructions of Local Filters for the Lipschitz Property

In this section, we prove Theorems 1.7, giving local filters of the Lipschitz property for functions $f : \mathcal{L}_n \to \mathbb{R}$ and $f : \mathcal{H}_{n,d} \to \mathbb{R}$. 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 1.7.

Definition 4.1 (Out-neighbors, outdegree)**.** *Given a directed graph $H = (V, E_H)$, let $\mathcal{N}_H(u)$ be the set $\{z \in V \mid (u, z) \in E_H\}$ of out-neighbors of vertex u in H and $\mathcal{N}^*_H(u) = \mathcal{N}_H(u) \cup \{u\}$. (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 $z \in \mathcal{N}^*(x) \cap \mathcal{N}^*(y)$ is on a shortest path between x and y in G.*
- *(Strict) Containment: $(x, y) \in E_H \implies \mathcal{N}(y) \subset \mathcal{N}(x)$.*

Lemma 4.1 (Lookup graph implies local Lipschitz filter)**.** *If 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 the 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. Recall that a function is Lipschitz on a set $D' \subseteq V(G)$ if it violates no pairs in $D' \times D'$. (See Definition 2.2.)

```
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 $\mathcal{N}^*(x)$, output $g(x) = f(x)$.
4  Otherwise, 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. We now prove a simple claim that we use further in the proof.

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 $|\mathcal{N}(x)| = n > 0$. Since each $z \in \mathcal{N}(x)$ has $|\mathcal{N}(z)| < n$, 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 the fact that $\mathcal{H}_{n,d}$ is a Cartesian graph product of d line graphs $\mathcal{L}_n$. Claim 4.4 shows that a strong product (Definition 4.4) of lookup graphs is a lookup graph of the product graph. We first define the Cartesian and strong graph products.

Definition 4.3 (Cartesian graph product)**.** *Given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the* Cartesian graph product, *denoted by $G_1 \times G_2$, is a graph with the vertex set $V_1 \times V_2$. It contains an edge from (x_1, x_2) to (y_1, y_2) if and only if $x_1 = y_1$ and $(x_2, y_2) \in E_2$, or $(x_1, y_1) \in E_1$ and $x_2 = y_2$.*

Definition 4.4 (Strong product)**.** *Given directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the* strong product *of G_1 and G_2, denoted $G_1 \Box G_2$, is a graph with the vertex set $V_1 \times V_2$ and the edge set*

$$\left\{((x_1, x_2), (y_1, y_2)) \mid (x_1, x_2) \in V_1 \times V_2, (y_1, y_2) \in \mathcal{N}^*_{G_1}(x_1) \times \mathcal{N}^*_{G_2}(x_2) \text{ and } (x_1, x_2) \neq (y_1, y_2)\right\}.$$

Claim 4.4. *Let G_1 and G_2 be undirected graphs with lookup graphs H_1 and H_2, respectively. Then the strong product $H = H_1 \Box H_2$ is a lookup graph of $G = G_1 \times G_2$. Moreover, $outdegree(H) \leq (outgedgree(H_1) + 1)(outdegree(H_2) + 1)$.*

Proof. Given $(u_1, u_2), (v_1, v_2) \in V_1 \times V_2$, let $p \in \mathcal{N}^*_{H_1}(u_1) \cap \mathcal{N}^*_{H_1}(v_1)$ (respectively, $q \in \mathcal{N}^*_{H_2}(u_2) \cap \mathcal{N}^*_{H_2}(v_2)$) be a vertex on a shortest path between u_1 and v_1 in G_1 (respectively, between u_2 and v_2 in G_2). (The vertices p and q exist by the *consistency* property of the lookup graphs.) Clearly, (p, q) lies in both $\mathcal{N}^*_H((u_1, u_2))$ and $\mathcal{N}^*_H((v_1, v_2))$. Since G is the Cartesian graph product of G_1 and G_2, (p, q) lies on a shortest path between (u_1, u_2) and (v_1, v_2) in G. This proves the *consistency* property of H. Now, fix an arbitrary vertex $(u_1, u_2) \in V_1 \times V_2$. By definition of the strong product, $((u_1, u_2), (x, y)) \in E_H$ if and only if $(x, y) \in \mathcal{N}^*_{H_1}(u_1) \times \mathcal{N}^*_{H_2}(u_2)$ and $(x, y) \neq (u_1, u_2)$. Therefore, $\mathcal{N}_H((u_1, u_2)) = \mathcal{N}^*_{H_1}(u_1) \times \mathcal{N}^*_{H_2}(u_2) \setminus \{(u_1, u_2)\}$. This immediately proves the *containment* property and the bound on $outdegree(H)$. □

Let H_n be the lookup graph for the line $\mathcal{L}_n$ constructed earlier. The lookup graph H for $\mathcal{H}_{n,d}$ is simply $\Box_{i=1}^d H_n = H_n \Box H_n \Box \cdots \Box H_n$, i.e., the strong product of H_n with itself taken $d - 1$ times. A simple induction and Claim 4.4 establishes the desired properties of H. This completes the proof of Lemma 4.3. □

Theorem 1.7 follows from Lemmas 4.1 and 4.3.

4.2 Lower bounds on local Lipschitz filters

In this section, we state and prove lower bounds on Lipschitz filters on the hypergrid graph, $\mathcal{H}_{n,d}$.

Theorem 4.5. *Consider a nonadaptive local Lipschitz filter with constant error probability δ. If the filter is for functions $f : \mathcal{H}_{n,d} \to \mathbb{R}$, it must perform $\Omega\Big(\frac{(\ln n-1)^{d-1}}{d(4\pi)^d}\Big)$ lookups per query. If the filter is for functions $f : \mathcal{H}_d \to \mathbb{R}$, it must perform $\Omega\left(2^{\alpha d}/d\right)$ lookups per query, where $\alpha \approx 0.1620$.*

Lemma 4.6 shows that a nonadaptive local Lipschitz filter for $\mathcal{H}_{n,d}$ with low lookup complexity gives a sparse 2-TC-spanner of the hypergrid. Together with the lower bounds on the size of 2-TC-spanners of the hypergrid and hypercube stated in Lemma 2.2, it implies Theorem 4.5.

Lemma 4.6. *A nonadaptive local Lipschitz filter on $G = \mathcal{H}_{n,d}$ with lookup complexity ℓ implies a 2-TC-Spanner of $\mathcal{H}_{n,d}$ with at most $n^d \ell \cdot \lceil 2d \log n / \log(1/2\delta) \rceil$ edges.*

Proof. Let $\mathcal{F} = \{x, y \in V : x \prec y\}$. Given $(x,y) \in \mathcal{F}$, let $\mathsf{cube}(x,y) = \{z \in V : x \preceq z \preceq y\}$. Now, for a fixed $(x,y) \in \mathcal{F}$, we define two functions f_1 and f_2 on V. Define $f_1(z) = d_G(x,z)$. Define $f_2(z) = d_G(x,z)$ for $z \notin \mathsf{cube}(x,y)$. For $z \in \mathsf{cube}(x,y)$, $f_2(z)$ is set to $d_G(x,z)+1$. It is clear that f_1 is Lipschitz on G and we prove shortly that f_2 is also Lipschitz on G. Call a random seed ρ good for $(x,y) \in \mathcal{F}$ if filter A_ρ on input f_1 outputs $g = f_1$ *and* on input f_2 outputs $g = f_2$, where f_1 and f_2 are functions defined with respect to (x,y) as above. Notice this happens with probability at least $1-2\delta$ by the union bound for any choice of the random seed. Let S be the set of s uniformly and independently chosen strings from the set of strings used as random seed by A. We claim that for every $(x,y) \in \mathcal{F}$ some string in S is good for (x,y) provided $s = \lceil 2d \log n / \log(1/2\delta) \rceil$. Specifically, since strings in S are chosen uniformly and independently, for any fixed (x,y), Pr[no string in S is *good* for (x,y)] $\le (2\delta)^s$. Then for $s = \log|\mathcal{F}| / \log(1/2\delta)$ this probability is at most $1/|\mathcal{F}|$. An application of the union bound and the fact that $|\mathcal{F}| \le |V|^2$ proves the claim about S.

Let A_ρ be the filter as in the statement of the lemma with the random seed fixed to ρ. Given $x \in V$, let $\mathcal{L}_\rho(x)$ be the set of lookups made by filter A_ρ on query x. We construct a 2-TC-spanner of G as follows: for each $\rho \in S$ and each $x \in V$, we add an edge between x and each comparable $u \in \mathcal{L}_\rho(x)$ and orient the edges according to $\overrightarrow{\mathcal{H}}_{n,d}$. (See Definition 2.3). We claim this gives a 2-TC-spanner of G. To prove this, we assume otherwise. That is, there exists $(x,y) \in \mathcal{F}$ with no path of length at most 2 in the constructed graph. Define functions f_1 and f_2 with respect to (x,y) as above and let $\rho \in S$ be the random seed which is *good* for (x,y). Define f_3 such that $f_3(z) = f_1(z)$ if $z \in \mathcal{L}_\rho(x)$ while $f_3(z) = f_2(z)$ if $z \in \mathcal{L}_\rho(y)$. On remaining points define f_3 arbitrarily. Function f_3 is well-defined because of the assumption that $\mathcal{L}_\rho(x)$ and $\mathcal{L}_\rho(y)$ do not intersect in c*ube*(x,y) and the fact that f_1 and f_2 are identical outside the set $\mathsf{cube}(x,y)$. Now, definition of f_3 implies $A_\rho(f_3, x) = 0$ and $A_\rho(f_3, y) = d_G(x,y)+1$. This is because ρ is *good* for (x,y) and the view of the filter on input x and y is identical to Lipschitz functions f_1 and f_2 respectively. But this means that the filter outputs a function which is not Lipschitz. Contradiction. It remains to show that f_2 is Lipschitz.

Consider any pair of vertices $z_1, z_2 \in V$. It is clear that f_2 may violate the pair $\{z_1, z_2\}$ only if one of the vertices is inside $\mathsf{cube}(x,y)$ and the other outside. Therefore, assume $z_1 \in \mathsf{cube}(x,y)$ and $z_2 \notin \mathsf{cube}(x,y)$. This implies $f_2(z_1) = d_G(x,z_1)+1$ and $f_2(z_2) = d_G(x,z_2)$. This immediately gives $f(z_2) - f(z_1) \le d_G(z_1,z_2)$ using triangle inequality. It remains to show that $f(z_2) - f(z_1) \ge -d_G(z_1,z_2)$, that is, $d_G(x,z_2) - d_G(x,z_1) - 1 \ge -d_G(z_1,z_2)$. An important observation is that for $x, y \in V$, $x \prec y$, a vertex z lies on a shortest path between x and y iff $z \in \mathsf{cube}(x,y)$. Since $z_2 \notin \mathsf{cube}(x,y)$, we get $d_G(x,z_2) + d_G(z_2,y) \ge d_G(x,y)+1$. Using the fact $z_1 \in \mathsf{cube}(x,y)$ and therefore $d_G(x,y) = d_G(x,z_1) + d_G(z_1,y)$, the last inequality simplifies to $d_G(x,z_2) + d_G(z_2,y) \ge d_G(x,z_1) + d_G(z_1,y) + 1$. Rearranging and applying the triangle inequality, we get $d_G(x,z_2) - d_G(x,z_1) - 1 \ge d_G(z_1,y) - d_G(z_2,y) \ge -d_G(z_1,z_2)$, as required.

The number of edges in H is at most

$$\sum_{x \in V, \rho \in S} |\mathcal{L}_\rho(x)| \leq n^d \cdot \ell \cdot s \leq n^d \ell \cdot \lceil 2d \log n / \log(1/2\delta) \rceil. \qquad \square$$

Remark 4.1. The only fact about the hypergrid $\mathcal{H}_{n,d} = (V, E)$ used in the proof of Lemma 4.6 is the following: For $x, y \in V$, $x \prec y$, a vertex z lies on a shortest path between x and y iff $z \in \mathsf{cube}(x, y)$. The proof holds for any graph satisfying this property. $\diamondsuit$

5 Application to Data Privacy

In Section 5.1, we review differential privacy and the Laplace mechanism from [23] and describe our filter mechanism. In Section 5.2, we instantiate the filter mechanism with the filter from Theorem 1.7 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, [23]). *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,* $\Pr[\mathcal{A}(x) \in S] \leq e^\epsilon \Pr[\mathcal{A}(y) \in S]$.

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 *Laplace mechanism* [23] is a randomized algorithm for evaluating functions on databases privately and efficiently.

Theorem 5.1 (Laplace Mechanism [23]). *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 $i \in [t]$.*

The Laplace mechanism adds noise proportional to a Lipschitz constant of the function f. Lipschitz filters provide an approach for releasing f privately and efficiently when a distrusted client supplies a Lipschitz constant c of f. If the client's claim about c is correct, this approach results in the same noise as the Laplace mechanism.

Theorem 5.2 (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: $\mathcal{A}^f_{Fil}(x) = c \cdot A(\frac{1}{c} \cdot f, x) + (Y_1, \ldots, Y_k)$, 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.

Proof. 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. Therefore, 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 An Instantiation of the Filter Mechanism for Histograms

Theorem 5.2 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 1.7, 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 1.7 in the filter mechanism to release functions $f : D \to \mathbb{R}$. The performance of the resulting algorithm is summarized in Corollary 5.3. 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 5.3 (Filter mechanism for histograms)**.** *Fix $c, \epsilon > 0$. For all functions $f : D \to \mathbb{R}$, the filter mechanism of Theorem 5.2 instantiated with the local filter of Theorem 1.7 is ϵ-differentially private and its running time is bounded by $(\log m)^{O(k)}$ evaluations of f. In addition, for c-Lipschitz functions f on D, the error of the mechanism, $\mathcal{E}(f, A_{Fil})$ is $O(c/\epsilon)$.*

Proof. Since two distinct databases $x, x' \in D = \{0, ..., m\}^k$ are neighbors iff the corresponding vertices in $\mathcal{H}_{m+1,k}$ are adjacent, it follows that the metric d_D on D is given by the shortest path metric on $\mathcal{H}_{m+1,k}$. Therefore, using the local Lipschitz filter from Theorem 1.7 in the filter mechanism of Theorem 5.2 to release functions $f : D \to \mathbb{R}$, we get the first part of the theorem. The claim about the running time follows from the running time of the local Lipschitz filter. For the second part, observe that the output of the filter mechanism for a c-Lipschitz function f on input $x \in D$ is exactly $f(x) + Lap(c/\epsilon)$. This is because the local filter of Theorem 5.2 has error probability 0. This implies $\mathcal{E}(f, A_{Fil}) = \sup_{x \in D} \mathbb{E}[|\text{Lap}(c/\epsilon)|] = c/\epsilon$, as required. □

Finally, we compare our filter mechanism with other known ϵ-differentially private mechanisms for releasing functions $f : D \to \mathbb{R}$, where D is a set of histograms over k-element universe with multiplicity m. We mentioned previously that, in general, computing the least Lipschitz constant of a given function is undecidable. However, for functions f over the hypergrid $\mathcal{H}_{m+1,k}$, it can be done by exhaustive search over all edges of $\mathcal{H}_{m+1,k}$ in time dominated by $O(m^k)$ evaluations of f. Therefore, our filter mechanism has the same error for an honest client and significantly better running time than the direct application of the Laplace mechanism.

Another point of comparison is the *noisy histogram* approach for releasing $f : D \to \mathbb{R}$ privately. One can release the histogram $x \in D$ using the Laplace mechanism and let the client apply f to the noisy histogram herself. Let $\mathcal{B}(x) = f \circ \mathcal{A}_{Lap}^{identity}(x)$ denote the resulting mechanism. We will show in Theorem 5.4 that for some functions f, this approach can result in expected error $\Omega(\sqrt{k}/\epsilon)$, even when f is Lipschitz. This is significantly worse than the expected error $\Theta(1/\epsilon)$ resulting from applying the filter mechanism to such a function.

Theorem 5.4. *There exist functions f such that releasing f results in expected error $\Omega(\sqrt{k}/\epsilon)$ with the noisy histogram approach, but only $O(1/\epsilon)$ with the filter mechanism.*

Proof. Given $S \subseteq [k]$, let $f_S : D \to \mathbb{R}$ be the function which on input $x \in D = \{0, ..., m\}^k$, outputs the sum of counts of each element in S: $f_S(x) = \sum_{i \in S} x_i$. We show that for each S with $|S| = \Omega(k)$, the error of the *noisy histogram* approach, $\mathcal{E}(f_S, \mathcal{B})$, is $\Omega(\sqrt{k}/\epsilon)$. In contrast, for each $S \subseteq [k]$, the error of the filter mechanism, $\mathcal{E}(f_S, \mathcal{A}_{Fil})$, is $O(1/\epsilon)$, by Corollary 5.3 and the fact that f_S is Lipschitz for each $S \subseteq [k]$.

On query f_S and database x, the noisy histogram approach outputs $\mathcal{B}(x) = \sum_{i \in [r]} (x_i + Y_i)$ where Y_i's are independently and identically distributed random variables in $Lap(1/\epsilon)$ and $|S| = r$. Denoting by Z_r the random variable $Y_1 + \ldots + Y_r$, we see that $\mathcal{E}(f_S, \mathcal{B}) = \sup_{x \in D} \mathbb{E}[|Z_r|]$. Let Bad denote the event $|Z_r| > \sqrt{r}/\epsilon$. Then, $\mathbb{E}[|Z_r|] \geq \mathbb{E}[|Z_r| \big| Bad] \cdot \Pr[Bad] \geq (\sqrt{r}/\epsilon) \cdot \Pr[Bad]$. Since $r = \Omega(k)$, it suffices to show that Bad occurs with constant probability. Towards this, let $b = 1/\epsilon$. Now, $\mathbb{E}[Z_r^4] = 12b^4 r(r+1) \leq 24b^4r^2 = 6(\mathbb{E}[Z_r^2])^2$. The inequality holds because (we may assume) r is at least 1 while the last equality holds because $\mathbb{E}[Z_r^2] = 2b^2 r$. Since, Z_r is symmetric about 0, using anti-concentration results of [29], restated below as Claim 5.5, we get, $\Pr[|Z_r| > \sqrt{r}/\epsilon] \geq 1/36$. (Specifically, by substituting $X = Z_r$, $\theta = 0$, $t = 1/\sqrt{2}$ and $\eta = 1/\sqrt{3}$ in the claim.) □

The following claim was used in the proof of Theorem 5.4.

Claim 5.5 (Fact 3.3, Proposition 3.7, [29])**.** *If a random variable X which is symmetric about 0 satisfies $\mathbb{E}[X] = 0$ and $\mathbb{E}[x^4] \leq (1/\eta')^4 \mathbb{E}[x^2]^2$, then for all $\theta \in \mathbb{R}$ and $0 < t < 1$, $\Pr[|X - \theta| > t\mathbb{E}[X^2]^{\frac{1}{2}}] \geq \eta^4(1 - t^2)^2$, where $\eta = \min(\eta', 1/\sqrt{3})$.*

Acknowledgment

The authors would like to thank Adam Smith and Grigory Yaroslavtsev for useful comments and discussions.

References

[1] N. Ailon and B. Chazelle. Information theory in property testing and monotonicity testing in higher dimension. *Inf. Comput.*, 204(11):1704–1717, 2006.

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

[3] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008.

[4] N. Alon and B. Schieber. Optimal preprocessing for answering on-line product queries. Technical Report 71/87, Tel-Aviv University, 1987.

[5] M. J. Atallah, K. B. Frikken, and M. Blanton. Dynamic and efficient key management for access hierarchies. In *ACM Conference on Computer and Communications Security*, pages 190–202, 2005.

[6] Bar-Yossef, Jayram, Kumar, and Sivakumar. An information statistics approach to data stream and communication complexity. *JCSS: Journal of Computer and System Sciences*, 68, 2004.

[7] T. Batu, R. Rubinfeld, and P. White. Fast approximate PCPs for multidimensional bin-packing problems. *Inf. Comput.*, 196(1):42–56, 2005.

[8] Y. Benyamini and J. Lindenstrauss. *Geometric nonlinear functional analysis. Vol. 1*, volume 48 of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 2000.

[9] P. Berman, A. Bhattacharyya, E. Grigorescu, S. Raskhodnikova, G. Yaroslavtsev, and D. P. Woodruff. Steiner transitive-closure spanners of d-dimensional posets. arXiv:1011.6100v1 [cs.DS], 2010.

[10] 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.

[11] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. P. Woodruff. Transitive-closure spanners. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 932–941, 2009.

[12] E. Blais, J. Brody, and K. Matulef. Property testing lower bounds via communication complexity. In *IEEE Conference on Computational Complexity*. IEEE Computer Society, 2011.

[13] H. L. Bodlaender, G. Tel, and N. Santoro. Tradeoffs in non-reversing diameter. *Nordic J. Comput.*, 1(1):111 – 134, 1994.

[14] J. Briët, S. Chakraborty, D. García-Soriano, and A. Matsliah. Monotonicity testing and shortest-path routing on the cube. In M. J. Serna, R. Shaltiel, K. Jansen, and J. D. P. Rolim, editors, *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 462–475. Springer, 2010.

[15] S. Chakraborty, E. Fischer, and A. Matsliah. Query complexity lower bounds for reconstruction of codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:67, 2010.

[16] A. K. Chandra, S. Fortune, and R. J. Lipton. Lower bounds for constant depth circuits for prefix problems. pages 109–117, 1983.

[17] A. K. Chandra, S. Fortune, and R. J. Lipton. Unbounded fan-in circuits and associative functions. pages 52–60, 1983.

[18] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving programs robust. PSU Technical Report, 2010.

[19] Chazelle and Seshadhri. Online geometric reconstruction. In *COMPGEOM: Annual ACM Symposium on Computational Geometry*, 2006.

[20] B. Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987.

[21] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.

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

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

[24] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *JCSS*, 60(3):717–751, 2000.

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

[26] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *STOC*, pages 474–483, 2002.

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

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

[29] P. Gopalan, R. O'Donnell, Y. Wu, and D. Zuckerman. Fooling functions of halfspaces under product distributions. In *IEEE Conference on Computational Complexity*, pages 223–234. IEEE Computer Society, 2010.

[30] S. Halevy and E. Kushilevitz. Testing monotonicity over graph products. In *ICALP*, pages 721–732, 2004.

[31] S. Kale, Y. Peres, and C. Seshadhri. Noise tolerance of expanders and sublinear expander reconstruction. In *FOCS*, pages 719–728. IEEE Computer Society, 2008.

[32] Kalyanasundaram and Schnitger. The probabilistic communication complexity of set intersection. *SIJDM: SIAM Journal on Discrete Mathematics*, 5, 1992.

[33] C. McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics*, pages 148–188, Cambridge, 1989. Cambridge University Press.

[34] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, 2010.

[35] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. ECCC Report TR04-010, 2004.

[36] S. Raskhodnikova. Transitive-closure spanners: A survey. In *Property Testing*, pages 167–196, 2010.

[37] A. A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci*, 106(2):385–390, 1992.

[38] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[39] I. Roy, H. E. Ramadan, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, 2010.

[40] R. Rubinfeld and A. Shapira. Sublinear time algorithms. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:13, 2011.

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

[42] M. E. Saks and C. Seshadhri. Parallel monotonicity reconstruction. In *Proceedings of the 19th Annual Symposium on Discrete Algorithms (SODA)*, pages 962–971, 2008.

[43] M. Thorup. Parallel shortcutting of rooted trees. *J. Algorithms*, 23(1):139–159, 1997.

[44] A. C.-C. Yao. Space-time tradeoff for answering range queries (extended abstract). In *STOC*, pages 128–136, 1982.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de