TESTING AND RECONSTRUCTION OF LIPSCHITZ FUNCTIONS WITH APPLICATIONS TO DATA PRIVACY*

MADHAV JHA AND SOFYA RASKHODNIKOVA†

Abstract. A function $f : D \to R$ is *Lipschitz* if $d_R(f(x), f(y)) \leq d_D(x, y)$ for all x, y in D, where d_R and d_D denote the distance metrics on the range and domain of f, respectively. We 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 differ from every function with the property on many values. A *local filter* reconstructs a desired property (in this case, Lipschitz) in the following sense: given an arbitrary function f and a query x, it returns $g(x)$, where the resulting function g satisfies the property, changing f only when necessary. If f has the property, g must be equal to f.

We design efficient testers and local reconstructors for functions over domains of the form $\{1, \ldots, n\}^d$, equipped with ℓ_1 distance, and give corresponding impossibility results. The algorithms we design have applications to program analysis and data privacy. The application to privacy is based on the fact that a function f of entries in a database of sensitive information can be released with noise of magnitude proportional to a Lipschitz constant of f, while preserving the privacy of individuals whose data is stored in the database (Dwork, McSherry, Nissim and Smith, TCC 2006). We give a differentially private mechanism, based on local filters, for releasing a function f when a purported Lipschitz constant of f is provided by a distrusted client.

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[1]. 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.) Intuitively, a Lipschitz constant of f is a bound on how sensitive f is to small changes in its input.

Lipschitz continuity[2] 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 [35]; in program analysis, it is considered as a measure of robustness to noise [17]; in data privacy, it is used to scale noise added to output $f(x)$ to preserve differential privacy of a database x [22]. 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* [45, 27] 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

*A preliminary version of this paper appeared in the 52nd Annual IEEE Symposium on Foundations of Computer Science [32].

†Pennsylvania State University, University Park, 16802, USA. Email: {mxj201,sofya}@cse.psu.edu. Supported by National Science Foundation (NSF/CCF CAREER award 0845701 and NSF grant CDI-0941553).

[1]More generally, we allow d_D to be a *quasimetric*, i.e., a function that satisfies all axioms of a metric, possibly except for symmetry. This generalization is used only in Section 3.2, where we consider the shortest path distance d_D on a directed graph. When the graph contains no path from a node u to a node v, the distance $d_D(u, v) = \infty$.

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

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). See Figure 1.1. Thus, A is computing with reconstructed data g instead of its original input f.

Local reconstruction [46] 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 [9], further requires that if the original input has the property, it should not be modified by the filter, i.e., if f has the property, g must be equal to f. Our application to data privacy has an unusual feature, not encountered in previous applications of filters: algorithm A needs to access its input only at one point x (corresponding to the database its holding). Nevertheless, we require *local filters*, not because of the distributed aspect they were initially developed for, but because when g depends on x, it might leak information about x and violate privacy.

Previous work on property testing and reconstruction. Property testing [27, 45] 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 [42, 44] 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 [23, 26, 21, 6, 25, 24, 30, 1, 10, 9, 13, 39, 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, 46, 9], convexity of points [19], graph expansion [33] and error-correcting codes [14]. The local model is addressed in [46, 14, 9], with only [46] providing local filters, and the other two papers focusing on lower bounds. Results on filters for properties other than monotonicity of functions do not seem directly relevant to our work.

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

the hypercube are the special cases of the hypergrid for $d = 1$ and $n = 2$, respectively, with vertices of the hypercube renumbered as $\{0,1\}^d$ instead of $\{1,2\}^d$.

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 [23, 21, 10, 24], the directed hypercube $\overrightarrow{\mathcal{H}}_d$ in [26, 21, 25, 13] and the directed hypergrid $\overrightarrow{\mathcal{H}}_{n,d}$ in [26, 21, 6, 30, 9, 2, 46], 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 $x \prec y$.)

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. We discuss it in more detail below, in the paragraph titled "Testing the Lipschitz property on the line".

1.1.1. Testing the Lipschitz property. 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.

Testing the Lipschitz property on the hypercube. The following theorem, proved in Section 3.1, gives 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 the image space of the function has low diameter.

DEFINITION 1.1 (Image diameter). *The image diameter of $f : D \to R$, denoted* $\mathrm{ImD}(f)$*, is* $\max_{x,y \in D} d_R(f(x), f(y))$.

THEOREM 1.2 (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 $O(\frac{d \cdot \min\{d, \mathrm{ImD}(f)\}}{\delta\epsilon})$ time for*[3] *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. (Observe that a function over the range $\{0, 1\}$ is always Lipschitz, so the tester for this case is trivial.) In general, the running time of our tester is $O(\frac{d^2}{\delta\epsilon})$.

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

[3] 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.

use to restore the Lipschitz property one dimension at a time[4]. 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 the average of the values on the endpoints and "rounding" it down and up to values in the range. 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. One of the main features of the averaging operator is that its action can be broken down 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. This allows us to prove the desired statement for the basic operator.

In Section 3.1, we obtain the following corollary for real-valued functions from Theorem 1.2 by discretizing their values.

COROLLARY 1.3. *There is a nonadaptive 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 $O(\frac{d \cdot \min\{d, \mathrm{ImD}(f)\}}{\delta\epsilon})$ time.*

We also give a lower bound on the query complexity of the tester for the hypercube which matches the upper bound in Theorem 1.2 for the case of the $\{0,1,2\}$ range, constant ϵ and $\delta = 1$.

THEOREM 1.4. *Every (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 $\{0,1,2\}$.*

We prove Theorem 1.4 in Section 3.1.3 using the method presented by Blais, Brody, and Matulef [11] of reducing a suitable communication complexity problem to the testing problem. In [11], this method is used 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 on $\overrightarrow{\mathcal{L}_n}$. Observe that the Lipschitz properties on $\mathcal{L}_n$ and on $\overrightarrow{\mathcal{L}_n}$ are identical. To see this, note that for all $x, y \in [n]$, such that $x < y$, the shortest path distance between x and y in $\mathcal{L}_n$ is $y - x$. For $\overrightarrow{\mathcal{L}_n}$, the shortest path distance from x to y is $y - x$, and from y to x is ∞. Therefore, both properties can be stated by requiring $|f(x) - f(y)| \leq y - x$ for all $x, y \in [n]$, such that $x < y$. We extend the monotonicity tester from [10], based on 2-transitive-closure spanners (see Definition 2.5), for functions $f : G_n \to \mathbb{R}$ by abstracting out the requirements on the property which are needed for the tester and the analysis to work.

[4]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 [26, 21]. 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 Boolean functions, 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 functions (in [21]) is proved by induction on the size of the range. We use the idea of repairing the function one dimension at a time. But new ideas are needed to make our analysis work. Observe that in the case of the Lipschitz property, Boolean functions are always Lipschitz, so there is nothing to test. In addition, for the Lipschitz property, not only the size of the range, but also the distances between points in the range play a role. Although 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.

Our tester works for any property $\mathcal{P}$ of a function $f : G_n \to R$, where R is an arbitrary range, provided that $\mathcal{P}$ satisfies the following requirements:

(a) $\mathcal{P}$ can be expressed in terms of conditions on pairs of domain points;
(b) the conditions in (a) are transitive: namely, for all $x \prec y \prec z$ in the domain[5], whenever (x, y) and (y, z) satisfy the above conditions, so does (x, z); and
(c) every 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 3.12.) Examples of edge-transitive properties include the c-Lipschitz property on directed hypergrids and variants of monotonicity. (See the discussion after Definition 3.12.)

The Lipschitz property for functions on $f : \overrightarrow{\mathcal{L}_n} \to R$ allows extension for most ranges R of interest. We characterize such ranges R (in Claim 3.14) as *discretely metrically convex* metric spaces. Metric convexity is a standard notion in geometric functional analysis (see, e.g. [7]). 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 1.5 (Definition 1.3 of [7] 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 $f : \overrightarrow{\mathcal{L}_n} \to R$, where R is discretely metrically convex.

THEOREM 1.6. *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 nonadaptively and with one-sided error in time $O(\frac{s(n)}{\epsilon n})$.*

Recall that the Lipschitz properties on $\mathcal{L}_n$ and $\overrightarrow{\mathcal{L}_n}$ are identical. We therefore get the following corollary.

COROLLARY 1.7. *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 (unweighted undirected) graphs $G = (V, E)$.*

When the range of f is $\mathbb{R}$ and the image diameter $ImD(f)$ is small, the following theorem, proved in Section 3.2.1, gives a faster tester than Corollary 1.7.

THEOREM 1.8. *The Lipschitz property of functions $f : \mathcal{L}_n \to \mathbb{R}$ can be tested nonadaptively and with one-sided error in time $O\big(\frac{\log \min\{n, \mathrm{ImD}(f)\}}{\epsilon}\big)$.*

Our next theorem, proved in Section 3.2.2, shows that the upper bound of Theorem 1.8 is tight for nonadaptive one-sided error testers. Even though it is stated for range $[r]$ for concreteness, as shown in Corollary 3.16 at the end of Section 3.2.2, it applies to all discretely metrically convex spaces which contain two points at distance r. In particular, this includes all the spaces listed in Corollary 1.7.

THEOREM 1.9. *Every nonadaptive one-sided error tester of the Lipschitz property of functions $f : \mathcal{L}_n \to [r]$ must make $\Omega(\log \min\{n, r\})$ queries.*

To prove Theorem 1.9, we construct a family of $\Omega(\log n)$ functions which are $1/4$-far from Lipschitz and have pairwise disjoint sets of violated pairs. Moreover, for every $r \in [n]$

[5] $\prec$ 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.

there are $\Omega(\log r)$ functions in the family with image diameter at most r. This enables us to prove the lower bound using Yao's principle. The construction of functions f in the family has a clean description in terms of the *discrete derivative* function Δf, defined by $\Delta f(1) = 0$ and $\Delta f(x) = f(x) - f(x-1)$ for all $x \geq 2$.

1.1.2. Reconstruction of the Lipschitz property. We present a local filter of the Lipschitz property for functions of the form $f : [n]^d \to \mathbb{R}$ which runs in time $(O(\log n))^d$. This result is stated in Theorem 1.10, which is proved in Section 4.1.

THEOREM 1.10 (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*. A lookup graph H is a directed acyclic graph with the same vertex set as the (undirected) domain graph G. A lookup graph H is *consistent* with G if every pair of vertices x and y in H have a common vertex z reachable from both x and y, such that z is a vertex on a shortest path between x and y in G. (A lookup graph is defined formally in Definition 4.2.) We show that the existence of a lookup graph implies a local Lipschitz filter where the lookup complexity of the filter is the maximum *reachable-degree* of a node in the lookup graph. The *reachable-degree* of a node in H is the size of the set of vertices reachable from the node in H. We obtain a lookup graph for $[n]$ with reachable-degree (of every node in H) 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 [9] 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 [17]. 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 [17] 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 [17].) 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.

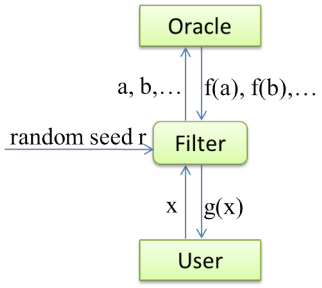


FIG. 1.1. *A property reconstructor: g always satisfies property P.*

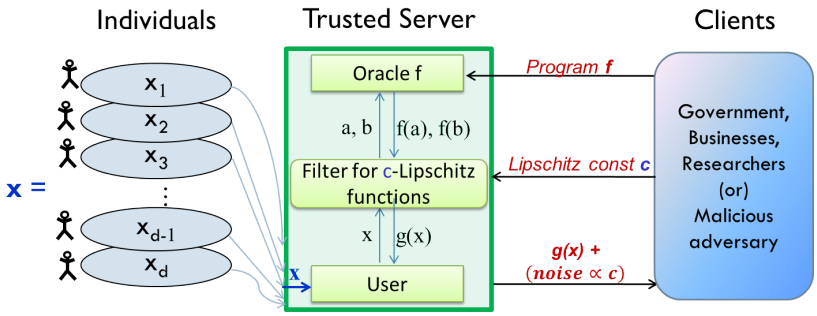


FIG. 1.2. *Use of a Lipschitz filter in private data analysis.*

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 about one individual. One of main questions addressed in this area is: what information about x that does not heavily depend on individual entries x_i can we compute (and release) efficiently? There is a vast body of work on this problem in statistics and computer science, with Dinur and Nissim [20] pioneering a line of work in cryptography. Dwork et al. [22] defined a rigorous notion of privacy, called *differential privacy* (reviewed in Defition 5.1), and described mechanisms, based on the global sensitivity (aka the Lipschitz constant), that achieve differential privacy for releasing a given function f of the database x. An example of such a mechanism is the Laplace mechanism, reviewed in Section 5.2. The method is based on adding random noise from a fixed distribution (e.g., 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.

Three major systems that release data while satisfying *differential privacy* have been implemented, all based on the Laplace mechanism[6]: PINQ [36], Airavat [43] and Fuzz [29]. All allow releasing functions of the database of the form $f : x \to \mathbb{R}$. In all implementations,

[6]Since the conference version of this paper was published, another system, GUPT [37], appeared. Instead of the Laplace mechanism, it is based on Sample and Aggregate [38]. It works for arbitrary functions f but is guaranteed to give accurate results only when $f(x)$ can be approximated well, based on random samples from x. Because of this restriction, this approach is incomparable to ours.

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 program f can be composed from a limited set of *trusted built-in* functions, such as sum and count. In addition, f can use a limited set of (untrusted) data transformations, such as applying a predicate to each row of the dataset, whose sensitivity can be enforced or deduced from the declared range of the transformation. PINQ and Airavat deduce the sensitivity of the overall program at runtime, while Fuzz checks it statically. The limitation of all three systems is that the functionality of the program is restricted either by the set of trusted built-in functions available (e.g. in PINQ and Airavat) or, in the case of Fuzz, the *expressivity* of the type systems.

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, 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 1.2.) Our mechanism can be instantiated with any privacy mechanism which is based on global sensitivity. The filter mechanism is differentially private and adds the same amount of noise for an honest client as the underlying privacy mechanism. 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[7]. Crucially, since the filter is local, it guarantees that g does not depend on the database x. That is, the client could have computed g by herself, based on f. Consequently, releasing $g(x)$ via the underlying privacy 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 underlying privacy mechanism.[8]

THEOREM 1.11 (Filter mechanism). *Let M be a privacy mechanism (e.g., the Laplace mechanism) whose inputs are a secret database $x \in D$, a positive constant c and a database query function $f : D \to R$, where c and f are supplied by a distrusted client. Suppose whenever f is c-Lipschitz, M is private (e.g., differentially private). Let F be* any *local Lipschitz filter satisfying Definition 2.1. The* filter mechanism *M' is identical to M, except that it uses $g(x) = c \cdot F(f/c, x)$ instead of $f(x)$ in all computations of M involving $f(x)$.*

The filter mechanism satisfies the following:

1. *The mechanism M' has the same privacy guarantee for the case when f and c are arbitrary as M has for the case when f is c-Lipschitz.*
2. *For honest clients, M' has the same error as M with probability $1 - \delta$ where δ is the error probability of F.*

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

[8] The definition of local filters we use (Definition 2.1), unlike the original one proposed by Saks and Seshadhri [46], does not require that f and g differ only on a small number of points. This requirement is unnecessary for the privacy application because the filter mechanism calls our filter only on one database x. If we added this requirement, a dishonest client would be penalized for fewer instances of x. Observe that the error that a filter introduces by substituting $f(x)$ with $g(x)$ does not depend on the distance of f to the Lipschitz property: it could be Lipschitz everywhere, besides x, but $f(x)$ would be changed anyway. However, it is not hard to see that our filter never changes $f(x)$ by more than $\max_y \{|f(y) - f(x)| + d_G(x, y)\}$.

3. *The increase in the running time of M' over M is bounded by the running time of F on input f and query x.*

Theorem 1.11 is proved in Section 5.1. We review the Laplace mechanism in Section 5.2. In Section 5.3, we instantiate the filter mechanism with the Laplace mechanism and our filter from Theorem 1.10 to obtain an efficient private algorithm for releasing real-valued functions f of the databases x.

Recall that the database x is modeled as a multiset (or a vector) over some domain U, where each element (resp., entry) $x_i \in U$ represents information about one individual. Our instantiation of the filter mechanism can be used when an upper bound on the multiplicity of all elements in the database is publicly known. (Note that the number of people in databases is a trivial upper bound.) When the client provides a correct Lipschitz constant, the resulting filter mechanism has the same expected error as the Laplace mechanism. Our mechanism is differentially private even for dishonest clients.

We show that when no reliable Lipschitz constant of f is given, previously known differentially private mechanisms (specifically, those based on the Laplace mechanism) either have a substantially higher running time (because they verify the Lipschitz constant by brute force) or have a higher expected error for a large class of functions f. Specifically, suppose that U has size k, that is, the individuals can have one of k *types*, and consider functions f that compute the number of individuals of types $S \subseteq [k]$ for $|S| = \Omega(k)$. We show that the *noisy histogram* approach (based on the Laplace mechanism) incurs an expected $\Omega(\sqrt{k}/\epsilon)$ error in answering the query. In contrast, our filter mechanism has expected error $O(1/\epsilon)$ while preserving differential privacy even in the presence of distrusted clients. The following theorem, proved in Section 5.3, summarizes the comparison of the filter mechanism to the noisy histogram approach.

THEOREM 1.12. *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.*

2. Preliminaries. In this section, we establish notation and review known results on property testing, local property reconstruction, Lipschitz functions and transitive-closure spanners.

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 [46], and the variant we consider was given in [9].

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 $g(x) = A_{f,\rho}(x)$ output by the filter must satisfy $\mathcal{P}$ for all f and ρ. In addition, if f satisfies $\mathcal{P}$ then g must be identical to f with probability at least $1 - \delta$ for some* error probability *$\delta \leq 1/3$, where the probability is taken over ρ.*

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

In [46], 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 a slowly growing function. We use the variant from [9] on this definition that does not have this requirement. (See Footnote 8.)

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 2.3 (Lemma 1.1, [7]). *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 3.12 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 2.4 (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. Recall that the *transitive closure* of a graph $G = (V, E)$ is a directed graph $TC(G)$ on the vertex set V such that there is an edge (x, y) in $TC(G)$ if and only if there is a directed path from x to y in G. Transitive-closure spanners (see [40] for a survey on the topic) are used in Sections 3.2.1 and 4.2.

DEFINITION 2.5 (k-TC-spanner, [10]). *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 [8, 9] 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 2.6. *A 2-TC-spanner of* $\overrightarrow{\mathcal{H}}_{n,d}$ *has* $\Omega\big(\frac{n^d(\ln n-1)^d}{(4\pi)^d}\big)$ *edges* [8]. *A 2-TC-spanner of* $\overrightarrow{\mathcal{H}}_d$ *has* $\Omega(2^{cd})$ *edges, where* $c \approx 1.1620$ [9].

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 1.2. Then we derive Corollary 1.3 on (a relaxation of) testing whether a function $f : \mathcal{H}_d \to \mathbb{R}$ is Lipschitz. At the end (in Section 3.1.3), we prove Theorem 1.4 which gives a lower bound on the query complexity of a Lipschitz tester on the hypercube.

When designing a tester of the Lipschitz property of functions $f : \mathcal{H}_d \to \delta\mathbb{Z}$, we may assume w.l.o.g. that $1/\delta$ is an integer. (This assumption is valid, more generally, when the domain is an undirected unweighted graph.) To see this, let $g : \mathcal{H}_d \to \mathbb{Z}$ be the function f/δ. Observe that g is $1/\delta$-Lipschitz if and only if for every edge $\{x, y\}$ in $\mathcal{H}_d$, the following holds: $|g(x) - g(y)| \leq 1/\delta$. Since g is an integer-valued function, the latter condition holds if and only if $|g(x)-g(y)| \leq \lfloor 1/\delta \rfloor$. Thus, g is $1/\delta$-Lipschitz iff g is $\lfloor 1/\delta \rfloor$-Lipschitz[9]. By rescaling, we get that f is Lipschitz iff $g/\lfloor 1/\delta \rfloor$ is 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 (Diameter). *The* diameter *of a metric space* (D, d_D)*, denoted* $diam(D)$*, is the maximum distance between any two points in* D*, i.e.,* $\max_{x,y\in D} d_D(x, y)$*. The* image diameter *of a function* $f : D \to \mathbb{R}$*, denoted* $\mathrm{ImD}(f)$*, is the difference between the maximum and the minimum values attained by* f*, i.e.,* $\max_{x\in D} f(x) - \min_{x\in D} f(x)$.

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* $V(f) \geq \delta\epsilon \cdot 2^{d-1}/\mathrm{ImD}(f)$.

Lemma 3.2 immediately yields a tester that works when (an upper bound on) the image diameter of function f is known. The following lemma, proved in Section 3.1.2, shows that knowing the image diameter is not necessary.

LEMMA 3.3 (Lipschitz tester oblivious to the image diameter). *Consider the Lipschitz property of functions* $f : D \to R$*, where* $R \subseteq \mathbb{R}$*. Suppose a one-sided error nonadaptive algorithm that* knows *an upper bound* r *on the image diameter* $\mathrm{ImD}(f)$ *of the input function can test this property in time* $T(\epsilon, r)$*. Then this property can be tested nonadaptively and with one-sided error in time*

$$2 \cdot T(\epsilon/2, \min\{diam(D), \mathrm{ImD}(f)\}) + O(1/\epsilon)$$

with no knowledge *of an upper bound on* $\mathrm{ImD}(f)$.

Proof of Theorem 1.2. In light of Lemma 3.3, it is sufficient to give a tester which knows an upper bound r on the image diameter of function f. The tester selects $s = \lceil 2dr/\delta\epsilon \rceil$ edges uniformly and independently at random from the hypercube $\mathcal{H}_d$. If any of the selected edges $\{x, y\}$ is violated, i.e., $|f(x) - f(y)| > 1$, it rejects; otherwise, it accepts.

[9] This does not hold for general domains. For example, consider the set $S = \{(0, 0), (0, 1), (1, 1)\} \subseteq \mathbb{R}^2$ and let function $f : S \to \mathbb{Z}$ be defined as follows: $f((0, 0)) = 0$, $f((0, 1)) = 1$ and $f((1, 1)) = 2$. Then, with respect to the ℓ_2-metric on the domain, f is $\sqrt{2}$-Lipschitz but **not** $\lfloor\sqrt{2}\rfloor$-Lipschitz.

The tester accepts all Lipschitz functions. Since the number of edges in the d-dimensional hypercube is $2^{d-1}d$, Lemma 3.2 implies that functions which are ϵ-far from Lipschitz are rejected with probability at least 2/3. The theorem (including the claim about the running time) then follows from Lemma 3.3. ■

Next we prove the corollary on a relaxation of testing if a function $f : \mathcal{H}_d \to \mathbb{R}$ is Lipschitz. We start by defining the rounding operators used in the proof of the corollary and later in Section 3.1.1.

DEFINITION 3.4 (Operators $\lfloor\ \rfloor_\delta$ and $\lceil\ \rceil_\delta$). *Let $\lfloor x \rfloor_\delta$ denote the largest value in $\delta\mathbb{Z}$ not greater than x. Similarly, let $\lceil x \rceil_\delta$ denote the smallest value in $\delta\mathbb{Z}$ not smaller than x.*

Proof of Corollary 1.3. 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 $x \in \{0,1\}^d$. Then $f(x) - \delta' \le f'(x) \le f(x)$ for all $x \in \{0,1\}^d$. If f is Lipschitz then f' is $(1+\delta')$-Lipschitz because $|f'(x) - f'(y)| \le |f(x) - f(y)| + \delta' \le 1 + \delta'$ 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 2.3.) Since $f'(x) \le f(x) \le f'(x) + \delta'$, function f is $(1 + 2\delta')$-Lipschitz on S, a contradiction.

Thus, we can use the Lipschitz tester of Theorem 1.2 with inputs $f'/(1+\delta'), d, \delta'/(1+\delta')$ and ϵ to distinguish Lipschitz f from 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 the average of the values on the endpoints and rounding it down and up to (almost) closest values in $\delta\mathbb{Z}$ to obtain new assignments for the endpoints. For the special case when $\delta = 1$, the rounding is in fact to the closest integer values: for every edge $\{x, y\}$ along dimension i, such that $f(x) < f(y) - 1$, we can define $A_i[f](x) = \lfloor \frac{f(x)+f(y)}{2} \rfloor$ and $A_i[f](y) = \lceil \frac{f(x)+f(y)}{2} \rceil$.

The definition of A_i for general δ is based on repeatedly applying the *basic operator* B_i, defined next. (This definition is equivalent to what we stated for $\delta = 1$, but does not directly generalize it.)

DEFINITION 3.5 (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 $B_i[f](y) = f(y) - \delta$.*

Now we define the *averaging operator* A_i.

DEFINITION 3.6 (Averaging operator A_i). *Given $f : \{0,1\}^d \to \delta\mathbb{Z}$, the* averaging operator A_i *applies B_i to the input function f multiple times until no edge along dimension i is violated.*

We can give another definition of the averaging operator, using the rounding operators from Definition 3.4. The new definition is equivalent to Definition 3.6, provided that $1/\delta$ is an integer. (As discussed in the beginning of Section 3.1, we can assume $1/\delta$ is an integer.) We give the second definition to present an alternative view of the operator, but do not actually use it in our arguments. Consequently, we omit the proof that the two definitions are equivalent.

DEFINITION 3.7 (Averaging operator A_i, equivalent definition). *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*

$$A_i[f](x) = \left\lceil \frac{f(x)+f(y)}{2} - \frac{1}{2} \right\rceil_\delta \text{ and } A_i[f](y) = \left\lfloor \frac{f(x)+f(y)}{2} + \frac{1}{2} \right\rfloor_\delta .$$

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

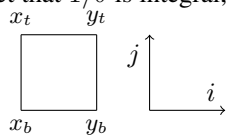
$$\delta V^i(f) \le VS^i(f) \le V^i(f) \cdot \mathit{ImD}(f). \tag{3.1}$$

Later, we use (3.1) 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 3.9. *For all* $i, j \in [d]$*, where* $i \neq j$*, and every function* $f : \{0, 1\}^d \to \delta\mathbb{Z}$*, applying the* averaging *operator* A_i *does not increase the violation score in dimension* j*, i.e.,* $VS_j(A_i[f]) \le VS_j(f)$*.*

Proof. Given Definition 3.6, in which the averaging operator is viewed as multiple applications of the basic operator, it suffices to prove Lemma 3.9 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 3.9 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, $f(y_t) \ge f(x_t) + 1 + \delta$. The application of the basic operator increases $f(x_t)$ by δ and decreases $f(y_t)$ by δ.

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

If the bottom edge is violated then, since $\mathbf{vs}(\{x_t, x_b\})$ increases and $1/\delta$ is integral, $f(x_t) \ge f(x_b) + 1 - \delta$. 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 $\frac{2}{\delta} \cdot V(f) \cdot \mathit{ImD}(f)$ points. We apply a sequence of averaging operators as follows: we define $f_0 = f$ and for all $i \in [d]$, we let $f_i = A_i[f_{i-1}]$. That is,

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

We claim that f_d is Lipschitz. By the 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 3.9, A_i preserves the Lipschitz property along dimensions fixed in the previous steps. Thus, eventually there are no violated edges, and f_d is Lipschitz.

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

$$\begin{aligned}\mathit{Dist}(f_{i-1}, f_i) = \mathit{Dist}(f_{i-1}, A_i[f_{i-1}]) \leq 2 \cdot V^i(f_{i-1}) \leq \frac{2}{\delta} \cdot VS^i(f_{i-1}) \leq \frac{2}{\delta} \cdot VS^i(f) \\ \leq \frac{2}{\delta} \cdot V^i(f) \cdot \mathit{ImD}(f). \qquad (3.2)\end{aligned}$$

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 (3.1). The third inequality holds because, by Lemma 3.9, the operators A_j for $j \neq i$ do not increase the violation score in dimension i. The distance from f to f_d is

$$\mathit{Dist}(f, f_d) \leq \sum_{i \in [d]} \mathit{Dist}(f_{i-1}, f_i) \leq \sum_{i \in [d]} \frac{2}{\delta} \cdot V^i(f) \cdot \mathit{ImD}(f) \qquad (3.3)$$

$$= \frac{2}{\delta} \cdot V(f) \cdot \mathit{ImD}(f). \qquad (3.4)$$

The two inequalities above follow from the triangle inequality and (3.2), respectively.

Consider a function f which is ϵ-far from the Lipschitz property. Since f_d is Lipschitz, $\mathit{Dist}(f, f_d) \geq \epsilon \cdot 2^d$. Together with (3.3), it gives $V(f) \geq \epsilon\delta \cdot 2^{d-1}/\mathit{ImD}(f)$, as required. ■

3.1.2. Analyzing the image diameter of a sample. In this section, we prove Lemma 3.3 which is used in the proofs of Theorems 1.2 and 1.8. First we give an algorithm for estimating the image diameter of a real-valued function. This is an important ingredient in the proof of Lemma 3.3.

CLAIM 3.10. *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 $r \leq \mathrm{ImD}(f)$ and with probability $\geq \frac{3}{4}$ the function f is ϵ-close to having image diameter at most r. Moreover, the algorithm runs in $O(1/\epsilon)$ time.*

Proof. We present Algorithm 1 (called SAMPLE-DIAMETER) and prove that it satisfies the requirements of Claim 3.10.

SAMPLE-DIAMETER always returns a number r no larger than $\mathit{ImD}(f)$. It remains to show that with probability $\geq \frac{3}{4}$, output r is such that the function f is ϵ-close to having image diameter at most r. To do that, 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 $x_1 = \mathrm{argmax}_{x \in L} f(x)$ and $x_2 = \mathrm{argmin}_{x \in R} f(x)$.

Algorithm 1 SAMPLE-DIAMETER$(f : D \to \mathbb{R}, \epsilon)$

1: Let $s = \lceil 5/\epsilon \rceil$.
2: Select samples $\mathbf{z} = z_1, \ldots, z_s$ from D uniformly and independently at random.
3: Return $r = \max_{i=1}^{s} f(z_i) - \min_{i=1}^{s} f(z_i)$.

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

$$\Pr(E_1 \cup E_2) \leq 2 \cdot \Pr(E_1) \leq 2 \cdot \left(1 - \frac{\epsilon}{2}\right)^{\lceil \frac{5}{\epsilon} \rceil} \leq 2 \cdot \left(e^{-\frac{\epsilon}{2}}\right)^{\frac{5}{\epsilon}} \leq \frac{1}{4}.$$

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

Proof of Lemma 3.3. The required tester is Algorithm 2.

Algorithm 2 TEST$(f : D \to R, \epsilon)$

1: Let $r \leftarrow$ SAMPLE-DIAMETER$(f, \epsilon/2)$. If $r > diam(D)$, **reject**.
2: Let $\mathcal{A}(f, \epsilon, r)$ be the Lipschitz tester as in Lemma 3.3. Namely, it gets oracle access to function $f : D \to R$ and receives a parameter r (in addition to ϵ) such that $r \geq \mathit{ImD}(f)$.
3: Run $\mathcal{A}(f, \epsilon/2, r)$ twice and **accept** if both runs of the algorithm accept; otherwise, **reject**.

Algorithm 2 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 Claim 3.10, $\Pr[E] \leq 1/4$. If $r > diam(D)$ then the tester correctly rejects on line 1 because, by Claim 3.10, $r \leq \mathit{ImD}(f)$, and a Lipschitz function on D must have image diameter at most $diam(D)$.

It remains to consider the case when $r \leq diam(D)$. Conditioned on E not happening (denoted $\overline{E}$), there is a function h with $\mathit{ImD}(h) \leq r$ such that $dist(f, h) < \epsilon/2$, where $dist(f, h)$ denotes the fraction of points on which f and h differ. In the following, assume E does not occur. Let $a_{min} = \min_{x \in D} h(x)$ and $a_{max} = \max_{x \in D} h(x)$. Consider a function g, obtained from f as follows:

$$g(x) = \begin{cases} a_{min} & \text{if } f(x) < a_{min}; \\ a_{max} & \text{if } f(x) > a_{max}; \\ f(x) & \text{otherwise.} \end{cases}$$

Then $\mathit{ImD}(g) \leq r$ and $dist(f, g) < \epsilon/2$. 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 has image diameter at most r and is $\epsilon/2$-far from Lipschitz, $\Pr[\mathcal{A}(g, \epsilon/2, r) \text{ rejects}] \geq 2/3$.

We claim that $\Pr[\mathcal{A}(f, \epsilon/2, r) \text{ rejects}] \geq \Pr[\mathcal{A}(g, \epsilon/2, r) \text{ rejects}]$. This is because, by construction of g, every pair violated by g is also violated by f. Since a nonadaptive 1-sided error tester rejects iff it queries a violated pair (x, y), we get that $\Pr[\mathcal{A}(f, \epsilon/2, r) \text{ rejects}] \geq \Pr[\mathcal{A}(g, \epsilon/2, r) \text{ rejects}] \geq 2/3$.

Since we run algorithm $\mathcal{A}$ twice and accept only if both runs accept, $\Pr[\text{Step 3 rejects } f \mid \bar{E}] \geq 1 - (1/3)^2 = 8/9$. Therefore,

$$\Pr[\text{Step 3 rejects } f] \geq \Pr[\text{Step 3 rejects } \mid \bar{E}] \cdot \Pr[\bar{E}] \geq \frac{8}{9} \cdot \frac{3}{4} = \frac{2}{3}.$$

Finally, observe that the running time is $O(1/\epsilon)$ if SAMPLE-DIAMETER returns $r > diam(D)$. Otherwise, it is $2 \cdot T(\epsilon/2, r)$, where $T(\epsilon/2, r)$ is the running time of $\mathcal{A}(f, \epsilon/2, r)$ and $r \leq \min\{diam(D), \mathit{ImD}(f)\}$. This completes the proof of Lemma 3.3. ■

3.1.3. Lower bound on the Lipschitz tester for the hypercube. In this section, we prove Theorem 1.4 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 [11] of reducing a suitable communication complexity problem to the testing problem.

Proof of Theorem 1.4. 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 [34, 5, 41] 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 3.11 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 1.4.

CLAIM 3.11. *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. Testing edge-transitive properties that allow extension. In this section, we prove Theorem 1.6, Corollary 1.7 and Theorem 1.8. We start by giving the definition and

examples of edge-transitive properties that allow extension. Recall the definition of comparable and incomparable elements (Definition 2.4).

DEFINITION 3.12. *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, fix such x and y and denote the distance from x to y in the underlying undirected graph by d. If we set $f(x) = 0$ and $f(y) = d+1$, there is no way to assign values of f on other vertices to ensure that f is Lipschitz, even though the current partial assignment violates no requirement on comparable vertices. 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 3.12. 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 Claim 3.14.

Next we prove Theorem 1.6 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 [10] for functions $f : G_n \to \mathbb{R}$, based on 2-TC-spanners (see Definition 2.5).

Proof of Theorem 1.6. 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 1.7 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 1.7. Since the Lipschitz properties on $\mathcal{L}_n$ and $\overrightarrow{\mathcal{L}_n}$ are identical, we prove the statement for the latter. 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 [48, 16, 15, 4, 18, 12, 47, 21]. It is surveyed as a stand-alone construction in [40]. We review this construction here, since it is used later, in the proof of Theorem 1.8. The edge set of the 2-TC-spanner is constructed recursively. In the construction, the middle node $v_{mid} = \lceil n/2\rceil$ is used as a hub: namely, we add edges (v, v_{mid}) for all nodes $v < v_{mid}$ and edges (v_{mid}, v) for all nodes $v > v_{mid}$. The construction then proceeds by recursively repeating the above procedure on the two line segments resulting from removing v_{mid} from the current line until each line segment contains exactly one node.

Since the Lipschitz property is edge-transitive and, by Claim 3.14, allows extension whenever R is discretely metrically convex, Theorem 1.6 implies the first part of the corollary. Then the second part, with the examples of spaces R, follows from Claim 3.15. ■

The strengthening of Corollary 1.7 to Theorem 1.8 for the case of range $\mathbb{R}$ and small image diameter is presented next. The improvement for this case stems from two observations. First, a function f with small image diameter cannot violate the Lipschitz condition on distant pairs of points. Second, for the real range, we can quickly estimate the image diameter from a small number of samples, as we already proved in Claim 3.10.

Proof of Theorem 1.8. We begin by proving the following claim.

CLAIM 3.13. *Let function $f : [n] \to \mathbb{R}$ be ϵ-far from Lipschitz. Let $H = ([n], E)$ be the 2-TC-spanner of the line graph constructed in the proof of Corollary 1.7. Let E' be the subset of E consisting of edges (x, y) satisfying $|x - y| < \mathrm{ImD}(f)$. Then the fraction of edges in E' violated by f is at least $\frac{\epsilon}{10 \log \min\{\mathrm{ImD}(f), n\}}$.*

Proof. From the proof of Theorem 1.6, we know that H has at least $\epsilon n/2$ violated edges. All these violated edges are contained in E', since every edge (x, y) with $|x - y| \geq \mathit{ImD}(f)$ must satisfy $|f(x) - f(y)| \leq \mathit{ImD}(f) \leq |x - y|$. Thus, E' has at least $\epsilon n/2$ violated edges.

It remains to show that $|E'| \leq 5n \log r$, where $r = \min\{\mathit{ImD}(f), n\}$. We say an edge $(x, y) \in E$ is of *length* j if $|x - y| = j$. There are at most $\log n$ recursive steps in the

construction of the 2-TC-spanner in the proof of Corollary 1.7. In the ith step, there are 2^{i-1} hubs, and at most $2r$ edges of length up to r per hub are added. Thus, in the first $\lceil \log(n/r) \rceil$ steps, at most $2r \cdot 2^{\lceil \log(n/r) \rceil} \leq 2r \cdot 2n/r = 4n$ edges are added to E'. In each of the remaining at most $\log n - \log(n/r) = \log r$ recursive steps, each node connects to at most one hub, that is, at most n edges are added to E'. Overall, $|E'| \leq 4n + n \log r \leq 5n \log r$. □

We first give a tester which works when the image diameter of f is known. The tester selects $s = \lceil 6 \log \min \{\mathit{ImD}(f), n\} / \epsilon \rceil$ edges uniformly and independently at random from the set E' defined in Claim 3.13. If any of the selected edges $\{x, y\}$ are violated, i.e., $|f(x) - f(y)| > 1$, it rejects; otherwise, it accepts. The tester accepts all Lipschitz functions. By Claim 3.13, functions which are ϵ-far from Lipschitz are rejected with probability at least 2/3. This tester, together with Lemma 3.3, implies the tester in the statement of Theorem 1.8, that works without the knowledge of *ImD*(f). ■

We finish this section by proving two claims used in the proof of Corollary 1.7. Claim 3.14 characterizes ranges R for which the Lipschitz property of functions $f : [n] \to R$ allows extension. Claim 3.15 gives examples of such ranges.

CLAIM 3.14. *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 3.15 (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 $d_{\ell_p}(\mathbf{u}, \mathbf{w}) \leq \alpha$ and $d_{\ell_p}(\mathbf{w}, \mathbf{v}) \leq \beta$. This shows that $(\mathbb{R}^k, \ell_p)$ for $p \in [1, \infty)$ is metrically convex. Fact 2.3 and Claim 3.14 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 $d_G(w,v) \le \beta$. 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 $d_G(u,w) + d_G(w,v) = d_G(u,v) \le \alpha + \beta$ implies $d_G(w,v) \le \beta$. 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 the 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 1.9.

Proof of Theorem 1.9. 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 one-sided error tester must accept if the input function is Lipschitz on the query points because, by Fact 2.3, 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 $f : [n] \to [r]$ which are 1/4-far from Lipschitz, and show that for every fixed query set $Q \subset [n]$ of size $o(\log \min\{n, r\})$, a function f drawn from $\mathcal{N}$ is Lipschitz on Q with probability more than 1/3.

0	1	1	1	2	1	1	0	0	-1	-1	-1	-2	-1	-1	0	0	1	1	1	2	1	1	0	0	-1	-1	-1	-2	-1	-1	0

FIG. 3.1. *The discrete derivative function,* Δf_i.

Fix positive integers $n, r \ge 8$. Assume $n = 2^\ell$. (We justify this assumption at the end of the proof.) For every $i \in [\ell - 2]$, we will give a function $f_i : [n] \to [2^{i+1}]$ which is $1/4$-far from Lipschitz. Let $t = \lfloor \log(\min\{n, r\}) \rfloor - 2$. Distribution $\mathcal{N}$ is uniform over functions in $\mathcal{F}_t = \{f_i : i \in [t]\}$. Observe every function in $\mathcal{F}_t$ has image diameter at most $2^{t+1} \le r$.

Fix $i \in [\ell - 2]$. We describe the function f_i by giving its *discrete derivative* function $\Delta f_i : [n] \to \mathbb{N}$, defined by $\Delta f(1) = 0$ and $\Delta f(x) = f(x) - f(x-1)$ for all $x \ge 2$. The value $f(x)$ for all $x \in [n]$ can be computed from the values of $\Delta f(x)$ as follows: $f_i(x) = 1 + \sum_{y\in[x]} \Delta f_i(y)$. The range of Δf_i is $\Sigma = \{-2, -1, 0, 1, 2\}$. To define Δf_i, consider the string σ over the alphabet Σ defined by

$$\sigma = (0 \;\; 1^{2^i-2} \;\; 1 \quad 2 \;\; 1^{2^i-2} \;\; 0 \quad 0 \;\; (-1)^{2^i-2} \;\; (-1) \quad (-2) \;\; 1^{2^i-2} \;\; 0)^{2^{\ell-i-2}},$$

where exponentiation denotes repetition, namely, a^b means a repeated b times. Then σ is of length $2^\ell = n$. Given $x \in [n]$, we define $\Delta f_i(x) = \sigma[x]$. (For an example when $n = 32$ and $i = 2$ see Figure 3.1.)

Now we show that the resulting functions f_i are 1/4-far from Lipschitz. 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 $j \in [n/2^i]$. 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.

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_s, a_{s+1}) is not violated for all $s \in [q-1]$. When f is chosen from $\mathcal{N}$, $\Pr[f \text{ violates } (a_s, a_{s+1})] \leq 1/|\mathcal{F}_t|$ for each $s \in [q-1]$. By the union bound,

$$\Pr[f \text{ is not Lipschitz on } Q] \leq \sum_{s \in [q-1]} \Pr[f \text{ violates } (a_s, a_{s+1})] \leq (q-1)/|\mathcal{F}_t|.$$

When $q \leq 2t/3$, this is less than 2/3. That is, every deterministic one-sided error nonadaptive test with q queries fails with probability greater than 1/3 on an input function drawn from $\mathcal{N}$. This completes the proof for the case when n is a power of 2.

If n is not a power of 2, let $\ell = \lfloor \log n \rfloor$. For every $i \in [\ell - 2]$, let $\tilde{f}_i$ be the function on $[n]$ such that it is identical to f_i for every $x \in [2^\ell]$. Moreover, for $x \geq \lfloor \log n \rfloor + 1$, $\tilde{f}_i(x)$ is defined to be $f_i(2^\ell)$. Since $\tilde{f}_i$ and f_i agree on at least 1/2 of the domain points, every $\tilde{f}_i$ is $1/8$-far from Lipschitz. Also $\tilde{f}_i$ has the same image diameter as f_i. Then all the arguments presented in the proof above still hold once we replace f_i with $\tilde{f}_i$ and correspondingly replace "1/4-far" with "1/8-far". ■

Next we generalize Theorem 1.9 to all discretely metrically convex spaces.

COROLLARY 3.16. *Let $r \in \mathbb{N}$. Consider a discretely metrically convex space R that contains two points at distance r. Every nonadaptive one-sided error tester of the Lipschitz property of functions $f : \mathcal{L}_n \to R$ must make $\Omega(\log \min\{n, r\})$ queries.*

The corollary follows from the following claim that states that the line $\mathcal{L}_r$ (used as the range space in Theorem 1.9) can be embedded into R without distortion if R is as stated in Corollary 3.16.

CLAIM 3.17. *Let $r \in \mathbb{N}$. Consider a discretely metrically convex space R that contains two points, u_1 and u_r, at distance r. Then R contains points $u_2, \ldots, u_{r-1}$ such that $d_R(u_i, u_{i+1}) = 1$ for all $i \in [r-1]$. (In other words, $\mathcal{L}_r$ can be embedded into R with no distortion by mapping i to u_i for all $i \in [r]$.)*

Proof. Let u_1 and u_r be two points in R at distance r. The proof is by induction on r. The base case, when $r = 2$, holds trivially. For the inductive case, assume $r > 2$ and suppose the claim holds for $r' = r - 1$. Since R is discretely metrically convex, by definition, there exists a point u_{r-1} such that $d_R(u_1, u_{r-1}) \leq r'$ and $d_R(u_{r-1}, u_r) \leq 1$. These inequalities are tight since otherwise, by triangle inequality, $d_R(u_1, u_r)$ would be less than r. Applying the induction hypothesis to the pair of points u_1 and u_{r-1}, we get the desired claim. □

4. Local reconstruction of the Lipschitz property.

4.1. Constructions of local filters for the Lipschitz property. In this section, we prove Theorems 1.10, giving local filters of the Lipschitz property for functions $f : \mathcal{L}_n \to \mathbb{R}$ and $f : \mathcal{H}_{n,d} \to \mathbb{R}$. Our filters are deterministic and nonadaptive. We abstract the combinatorial object used in these filters as a *lookup graph*. We start by defining lookup graphs in Definition 4.2. In Lemma 4.3, we show how to use them to construct Lipschitz filters. Finally, we construct lookup graphs for the line and the hypergrid in Lemma 4.5. Lemmas 4.3 and 4.5 imply Theorem 1.10.

DEFINITION 4.1. *Consider a directed acyclic graph $H = (V, E_H)$ and a node $x \in V$. Let $\mathcal{N}_H(x)$ be the set $\{y \in V \mid (x, y) \in E_H\}$ of out-neighbors of x in H. Let $\mathcal{R}_H(x)$ be the set of nodes (other than x) reachable from x in H. Also, let $\mathcal{N}^*_H(x) = \mathcal{N}_H(x) \cup \{x\}$ and $\mathcal{R}^*_H(x) = \mathcal{R}_H(x) \cup \{x\}$. (We omit the subscript H when the graph is clear from the context.)*

We denote the maximum outdegree of a node in H by out-deg(H). Finally, reach-deg$(H) = \max_{x \in V(H)} |\mathcal{R}_H(x)|$.

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

- CONSISTENCY *(of sets $\mathcal{R}_H(x)$): for all $x, y \in V$, some $z \in \mathcal{R}^*_H(x) \cap \mathcal{R}^*_H(y)$ is on a shortest path between x and y in G.*
- PROXY *(sets $\mathcal{N}_H(x)$ are proxies for sets $\mathcal{R}_H(x)$): for all $x \in V$ and $y \in \mathcal{R}_H(x)$, some $z \in \mathcal{N}_H(x)$ is on a shortest path between x and y in G.*

LEMMA 4.3 (Lookup graph implies local Lipschitz filter). *If a graph G has a lookup graph H then there is a nonadaptive local Lipschitz filter for real-valued functions on G with lookup complexity reach-deg(H) and running time $O($reach-deg$(H) \cdot$ out-deg$(H))$.*

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

Algorithm 3 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 $g(z) =$ 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 3 terminates because H is acyclic. Function g returned by the filter is identical to the input function f, provided that f is Lipschtiz.

We now prove a simple claim used to show that function g is Lipschitz.

CLAIM 4.4. *If function f is Lipschitz on $\mathcal{R}^*(x)$ and on $\mathcal{R}^*(y)$, it is also Lipschitz on $\{x, y\}$.*

Proof. Let $z \in \mathcal{R}^*(x) \cap \mathcal{R}^*(y)$ 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 $\{x, y\}$. □

To show that function g returned by Algorithm 3 is Lipschitz, it is sufficient, by Claim 4.4, to prove that for each $x \in V$, function g is Lipschitz on $\mathcal{R}^*(x)$. The proof is by strong induction on $|\mathcal{R}(x)|$. The base case (when $|\mathcal{R}(x)| = 0$) holds for trivial reasons. For the inductive case, let $|\mathcal{R}(x)| = k > 0$. Since each $z \in \mathcal{R}(x)$ has $|\mathcal{R}(z)| < k$, the induction hypothesis gives us that g is Lipschitz on $\mathcal{R}^*(z)$ for all $z \in \mathcal{R}(x)$. Then Claim 4.4 implies that g is Lipschitz on $\mathcal{R}(x)$. Lines 3 and 4 in Algorithm 3 ensure that g is Lipschitz on $\mathcal{N}^*(x)$. By the *proxy* property of H, function g is then Lipschitz on $\mathcal{R}^*(x)$, as required.

On query x, the filter only looks up nodes reachable from x. Therefore, the lookup complexity of the filter is at most reach-deg(H). Moreover, if for all nodes x, the filter stores the value of $g(x)$ the first time it is computed and reuses it later, the running time is $O($reach-deg$(H) \cdot$ out-deg$(H))$. This is because the number of recursive calls to the filter is at most reach-deg(H) and the time spent on each call is $O($out-deg$(H))$. □

LEMMA 4.5 (Lookup graph constructions). *The line graph $\mathcal{L}_n$ has a lookup graph H with out-deg$(H) = 2$ and reach-deg$(H) = O(\log n)$. The hypergrid $\mathcal{H}_{n,d}$ has a lookup graph H with out-deg$(H) = 3^d$ and reach-deg$(H) = (O(\log n))^d$.*

Proof. To construct a lookup graph H for the line $\mathcal{L}_n$, consider a balanced (rooted) *binary search tree* T on the set $[n]$. Each element x of $[n]$ can be viewed as a node of T and as an integer. Let $\ell a(x)$ be the largest ancestor of x in T which is smaller than x. Analogously, let $ra(x)$ be the smallest ancestor of x which larger than x. For every $x \in [n]$, add the edge $(x, \ell a(x))$ to H if $\ell a(x)$ exists and add the edge $(x, ra(x))$ to H if $ra(x)$ exists.

The resulting graph H is a DAG because all its edges go from nodes to their ancestors. Next we show that H satisfies the consistency and the proxy properties of Definition 4.2. Observe that for each node x other than the root, either $\ell a(x)$ or $ra(x)$ is the parent of x in T, so in H each x has an outgoing edge to its parent. Therefore, the set $\mathcal{R}_H(x)$ is the set of all ancestors of x in T. Recall that the lowest common ancestor (LCA) of vertices x, y in T is a common ancestor of x and y which is furthest from the root. For all distinct $x, y \in [n]$, the vertex $LCA(x, y)$ is reachable from both x and y in H and, by the binary search tree property, lies on the shortest path between x and y in $\mathcal{L}_n$, that is, $x \leq LCA(x, y) \leq y$. Thus, H satisfies the consistency property. To see that it also satisfies the proxy property, consider $x \in [n]$ and $y \in \mathcal{R}(x)$. Then y is an ancestor of x in T. If $y < x$ then $y < \ell a(x) \leq x$; if $x < y$ then $x < ra(x) \leq y$. In either case, some out-neighbor of x is on the shortest path between x and y in $\mathcal{L}_n$. This verifies that H is a lookup graph of $\mathcal{L}_n$.

By construction, out-deg$(H) = 2$. Since the binary search tree T is balanced, each vertex has $O(\log n)$ ancestors. Hence, reach-deg$(H) = O(\log n)$.

To construct a lookup graph for $\mathcal{H}_{n,d}$, we use the fact that $\mathcal{H}_{n,d}$ is the Cartesian product of d line graphs $\mathcal{L}_n$. Claim 4.9 shows that the strong product (Definition 4.7) of lookup graphs is a lookup graph of the Cartesian product graph. We first define the Cartesian and strong graph products.

DEFINITION 4.6 (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 4.7 (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 $G_1 \square G_2$, is a graph with the vertex set $V_1 \times V_2$ and the edge set*

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

We use the following fact about shortest paths in Cartesian graph products to prove Claim 4.9.

FACT 4.8. *Let $G = G_1 \times G_2$ be the Cartesian graph product of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. For each $i \in \{1, 2\}$, consider vertices $x_i, y_i, z_i \in V_i$, where z_i lies on a shortest path between x_i and y_i in G_i. Then vertex (z_1, z_2) lies on a shortest path between (x_1, x_2) and (y_1, y_2) in G.*

CLAIM 4.9. *Let G_1 and G_2 be undirected graphs with lookup graphs H_1 and H_2, respectively. Then the strong product $H = H_1 \square H_2$ is a lookup graph of $G = G_1 \times G_2$. Moreover, out-deg$(H) \leq$ (out-deg$(H_1) + 1$)(out-deg$(H_2) + 1$) and reach-deg$(H) \leq$ (reach-deg$(H_1) + 1$)(reach-deg$(H_2) + 1$).*

Proof. The strong product of two DAGs is a DAG. Next we prove that H satisfies the two properties of Definition 4.2. Given vertices $(x_1, x_2), (y_1, y_2)$ of G, let $z_1 \in \mathcal{R}^*_{H_1}(x_1) \cap \mathcal{R}^*_{H_1}(y_1)$ be a vertex on a shortest path between x_1 and y_1 in G_1, whose existence is guaranteed by the consistency property of H_1. Define z_2 analogously. Since $H = H_1 \square H_2$, vertex

(z_1, z_2) is in both $\mathcal{R}^*_H((x_1, x_2))$ and $\mathcal{R}^*_H((y_1, y_2))$. By Fact 4.8, (z_1, z_2) lies on a shortest path between (x_1, x_2) and (y_1, y_2) in G. This proves the *consistency* property of H.

To prove the proxy property, consider a vertex $x = (x_1, x_2)$ in G and a vertex $y = (y_1, y_2)$ in $\mathcal{R}_H(x)$. Observe that $\mathcal{R}_H(x) = \mathcal{R}^*_{H_1}(x_1) \times \mathcal{R}^*_{H_2}(x_2) \setminus \{x\}$. If $y_1 \in \mathcal{R}_{H_1}(x_1)$, let $z_1 \in \mathcal{N}_{H_1}(x_1)$ be a vertex on a shortest path between x_1 and y_1 in G_1, whose existence is guaranteed by the consistency property of H_1. Otherwise (if $x_1 = y_1$), let $z_1 = x_1$. Define z_2 analogously. Then (z_1, z_2) is in $\mathcal{N}_H((x_1, x_2))$ and, by Fact 4.8, it also lies on a shortest path between (x_1, x_2) and (y_1, y_2). This proves the proxy property.

The claimed bounds on out-deg(H) and reach-deg(H) follow from the fact that $\mathcal{N}^*_H(x) = \mathcal{N}^*_{H_1}(x_1) \times \mathcal{N}^*_{H_2}(x_2)$ and $\mathcal{R}^*_H(x) = \mathcal{R}^*_{H_1}(x_1) \times \mathcal{R}^*_{H_2}(x_2)$ for all nodes $x = (x_1, x_2)$ of 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 $\square_{i=1}^{d} H_n = H_n \square H_n \square \cdots \square H_n$, i.e., the strong product of H_n with itself taken $d-1$ times. A simple induction and Claim 4.9 establish the desired properties of H. This completes the proof of Lemma 4.5. □

Theorem 1.10 follows from Lemmas 4.3 and 4.5.

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 4.10. *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 $\Omega(\frac{(\ln n - 1)^{d-1}}{d(4\pi)^d})$ lookups per query. If the filter is for functions $f : \mathcal{H}_d \to \mathbb{R}$, it must perform $\Omega(2^{\alpha d}/d)$ lookups per query, where $\alpha \approx 0.1620$.*

Lemma 4.11 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 2.6, it implies Theorem 4.10.

LEMMA 4.11. *A nonadaptive local Lipschitz filter on $\mathcal{H}_{n,d}$ with lookup complexity ℓ implies a 2-TC-spanner of $\overrightarrow{\mathcal{H}}_{n,d}$ with at most $n^d \ell \cdot \lceil 4d \log n / \log(1/2\delta) \rceil$ edges.*

Proof. Let $G = (V, E)$ be the hypergrid $\mathcal{H}_{n,d}$. Consider the directed hypergrid $\overrightarrow{\mathcal{H}}_{n,d}$ on the same vertex set. Let $\mathcal{F} = \{x, y \in V : x \prec y\}$. (The partial order $\prec$ is defined in Definition 2.4.) 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:

$$f_1(z) = d_G(x, z);$$
$$f_2(z) = \begin{cases} d_G(x, z) & \text{for } z \notin \mathsf{cube}(x, y); \\ d_G(x, z) + 1 & \text{for } z \in \mathsf{cube}(x, y). \end{cases}$$

It is clear that f_1 is Lipschitz on G, and we prove shortly that f_2 is also Lipschitz on G.

Let A be a filter as in the statement of the lemma, and let A_ρ be filter A run with the random seed fixed to ρ. 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 above with respect to (x, y). By the union bound, a random seed is good for a given pair (x, y) with probability at least $1 - 2\delta$. We claim that there exists a (small) set S of random seeds used by the filter, such that for every $(x, y) \in \mathcal{F}$, some string in S is good for (x, y). To show this, we use a probabilistic argument from [9]. Specifically, let S be a set of $s = \lceil 2 \log |\mathcal{F}| / \log(1/2\delta) \rceil$ strings chosen uniformly and independently from the set of random seeds used by A. Since strings in S are chosen uniformly and independently, for any fixed

(x, y), $\Pr[\text{no string in } S \text{ is } \textit{good} \text{ for } (x, y)] \leq (2\delta)^s < 1/|\mathcal{F}|$. An application of the union bound proves the claim about S. Since $|\mathcal{F}| \leq |V|^2$, we get $s \leq \lceil 4d \log n / \log(1/2\delta) \rceil$.

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 $\overrightarrow{\mathcal{H}}_{n,d}$. (Comparable vertices are defined in Definition 2.4). We claim this gives a 2-TC-spanner of G. Towards contradiction, assume otherwise. That is, there exists $(x, y) \in \mathcal{F}$ with no path of length at most 2 in the constructed graph. This can happen only if $\mathcal{L}_\rho(x) \cap \mathcal{L}_\rho(y) \cap \mathsf{cube}(x, y) = \emptyset$. 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 the 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 $\mathsf{cube}(x, y)$ and the fact that f_1 and f_2 are identical outside the set $\mathsf{cube}(x, y)$. The 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 query x (respectively, y) is identical to the same view when the input is the Lipschitz function f_1 (respectively, f_2). But this means that the filter outputs a function which violates the pair (x, y) and hence is not Lipschitz. Contradiction. In the proof above, we claimed function f_2 is Lipschitz. We prove this next.

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 $f(z_2) - f(z_1) \leq d_G(z_1, z_2)$ using triangle inequality. It remains to show that $f(z_2) - f(z_1) \geq -d_G(z_1, z_2)$, that is, $d_G(x, z_2) - d_G(x, z_1) - 1 \geq -d_G(z_1, z_2)$. 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 $d_G(x, z_2) + d_G(z_2, y) \geq d_G(x, y) + 1$. 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 $d_G(x, z_2) + d_G(z_2, y) \geq d_G(x, z_1) + d_G(z_1, y) + 1$. Rearranging and applying the triangle inequality, we get

$$d_G(x, z_2) - d_G(x, z_1) - 1 \geq d_G(z_1, y) - d_G(z_2, y) \geq -d_G(z_1, z_2),$$

as required.

The number of edges in H is at most

$$\sum_{x \in V, \rho \in S} |\mathcal{L}_\rho(x)| \leq n^d \cdot \ell \cdot s \leq n^d \ell \cdot \lceil 4d \log n / \log(1/2\delta) \rceil. \qquad \square$$

REMARK 4.12. *The only fact about the hypergrid $\mathcal{H}_{n,d} = (V, E)$ used in the proof of Lemma 4.11 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.*

5. Application to data privacy. In Section 5.1, we analyze the performance of the filter mechanism, proving Theorem 1.11. In Section 5.2, we review differential privacy and the Laplace mechanism from [22]. In Section 5.3, we instantiate the filter mechanism with the Laplace mechanism and the filter from Theorem 1.10 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. Here we prove Theorem 1.11, stated in Section 1.2, which summarizes the performance of the *filter mechanism*.

Proof of Theorem 1.11. Let x be the input database and g_ρ be the output function of the local Lipschitz filter F with input function f/c and 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 M' can thus be seen as an application of the mechanism M on the c-Lipschitz function $c \cdot g_\rho$. Therefore, M' has the same privacy guarantees as M. Since ρ was arbitrary, this analysis holds for any choice of ρ, i.e., any instantiation of the filter F.

For the second part of the theorem, note that if f is c-Lipschitz, the function that filter F 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.

The final part about the running time of the mechanism follows from the definition of the filter mechanism. ■

5.2. Review of the Laplace 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*, i.e., a vector where the ith component represents the number of times the ith element of U occurs in the database. (Such representation is considered, e.g., in [31].) 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, [22]). *Fix $\epsilon > 0$. A randomized algorithm $\mathcal{A}$ is ϵ-differentially private if for all neighbors $x, y \in D$, and for all subsets S of outputs,* $\Pr[\mathcal{A}(x) \in S] \leq e^\epsilon \Pr[\mathcal{A}(y) \in S]$.

Recall that $Lap(\lambda)$ denote the Laplace distribution on $\mathbb{R}$ with the *scale parameter* λ. The *Laplace mechanism* [22] is a randomized algorithm for evaluating functions on databases privately and efficiently.

THEOREM 5.2 (Laplace Mechanism [22]). *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 c of the function f. Lipschitz filters provide an approach for releasing f privately and efficiently when a distrusted client supplies c. When this mechanism is instantiated with the Laplace mechanism, if the client's claim about c is correct, this approach results in the same noise as the Laplace mechanism itself.

5.3. An instantiation of the filter mechanism for histograms. Theorem 1.11 applies to arbitrary metric spaces (D, d_D). In this section, we instantiate the filter mechanism with the Laplace mechanism and with the local Lipschitz filter for functions from the hypergrid to real numbers, described in Theorem 1.10, 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 1.10 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 1.11 instantiated with the Laplace mechanism and the local filter of Theorem 1.10 is ϵ-differentially private and its running time is bounded by $(\log(m+1)+1)^k$ evaluations of f. In addition, for c-Lipschitz functions f on D, the error of the mechanism, $\mathcal{E}(f, A_{Fil})$ is $O(c/\epsilon)$.*

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 1.10 in the filter mechanism of Theorem 1.11 instantiated with the Laplace mechanism to release functions $f : D \to \mathbb{R}$, we get the first part of the corollary. 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 1.11 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. In Theorem 1.12, we show 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.

Proof of Theorem 1.12. 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, $\mathbb{E}[|Z_r|] \geq \mathbb{E}[|Z_r||Bad] \cdot \Pr[Bad] \geq (\sqrt{r}/\epsilon) \cdot \Pr[Bad]$. Since $r = \Omega(k)$, it suffices to show that Bad occurs with

constant probability. Towards this, let $b = 1/\epsilon$. Now, $\mathbb{E}[Z_r^4] = 12b^4r(r+1) \leq 24b^4r^2 = 6(\mathbb{E}[Z_r^2])^2$. The inequality holds because (we may assume) r is at least 1 while the last equality holds because $\mathbb{E}[Z_r^2] = 2b^2r$. Since, Z_r is symmetric about 0, using anti-concentration results of [28], restated below as Claim 5.4, 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 1.12.

CLAIM 5.4 (Fact 3.3, Proposition 3.7, [28]). *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] NIR AILON AND BERNARD CHAZELLE, *Information theory in property testing and monotonicity testing in higher dimension.*, Inf. Comput., 204 (2006), pp. 1704–1717.

[2] NIR AILON, BERNARD CHAZELLE, SESHADHRI COMANDUR, AND DING LIU, *Estimating the distance to a monotone function*, Random Struct. Algorithms, 31 (2007), pp. 371–383.

[3] ———, *Property-preserving data reconstruction*, Algorithmica, 51 (2008), pp. 160–182.

[4] NOGA ALON AND BARUCH SCHIEBER, *Optimal preprocessing for answering on-line product queries*, Tech. Report 71/87, Tel-Aviv University, 1987.

[5] ZIV BAR-YOSSEF, T. S. JAYRAM, RAVI KUMAR, AND D. SIVAKUMAR, *An information statistics approach to data stream and communication complexity*, J. Comput. Syst. Sci., 68 (2004), pp. 702–732.

[6] TUGKAN BATU, RONITT RUBINFELD, AND PATRICK WHITE, *Fast approximate PCPs for multidimensional bin-packing problems*, Inf. Comput., 196 (2005), pp. 42–56.

[7] YOAV BENYAMINI AND JORAM LINDENSTRAUSS, *Geometric nonlinear functional analysis. Vol. 1*, vol. 48 of Colloquium Publications, Amer. Math. Soc., 2000.

[8] PIOTR BERMAN, ARNAB BHATTACHARYYA, ELENA GRIGORESCU, SOFYA RASKHODNIKOVA, DAVID P. WOODRUFF, AND GRIGORY YAROSLAVTSEV, *Steiner transitive-closure spanners of low-dimensional posets*, in ICALP (1), Luca Aceto, Monika Henzinger, and Jiri Sgall, eds., vol. 6755 of Lecture Notes in Computer Science, Springer, 2011, pp. 760–772.

[9] ARNAB BHATTACHARYYA, ELENA GRIGORESCU, MADHAV JHA, KYOMIN JUNG, SOFYA RASKHODNIKOVA, AND DAVID P. WOODRUFF, *Lower bounds for local monotonicity reconstruction from transitive-closure spanners*, SIAM Journal on Discrete Mathematics, 26 (2012), pp. 618–646.

[10] ARNAB BHATTACHARYYA, ELENA GRIGORESCU, KYOMIN JUNG, SOFYA RASKHODNIKOVA, AND DAVID P. WOODRUFF, *Transitive-closure spanners*, SIAM J. Comput., 41 (2012), pp. 1380–1425.

[11] ERIC BLAIS, JOSHUA BRODY, AND KEVIN MATULEF, *Property testing lower bounds via communication complexity*, Computational Complexity, 21 (2012), pp. 311–358.

[12] HANLS L. BODLAENDER, GERARD TEL, AND NICOLA SANTORO, *Tradeoffs in non-reversing diameter*, Nordic J. Comput., 1 (1994), pp. 111 – 134.

[13] JOP BRIËT, SOURAV CHAKRABORTY, DAVID GARCÍA-SORIANO, AND ARIE MATSLIAH, *Monotonicity testing and shortest-path routing on the cube*, Combinatorica, 32 (2012), pp. 35–53.

[14] SOURAV CHAKRABORTY, ELDAR FISCHER, AND ARIE MATSLIAH, *Query complexity lower bounds for reconstruction of codes*, in 2nd Symposium on Innovations in Computer Science (ICS), 2011.

[15] ASHOK K. CHANDRA, STEVEN FORTUNE, AND RICHARD J. LIPTON, *Lower bounds for constant depth circuits for prefix problems*, in ICALP, Josep Díaz, ed., vol. 154 of Lecture Notes in Computer Science, Springer, 1983, pp. 109–117.

[16] ———, *Unbounded fan-in circuits and associative functions*, J. Comput. Syst. Sci., 30 (1985), pp. 222–234.

[17] SWARAT CHAUDHURI, SUMIT GULWANI, ROBERTO LUBLINERMAN, AND SARA NAVIDPOUR, *Proving programs robust*, in Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ESEC/FSE '11, New York, NY, USA, 2011, ACM, pp. 102–112.

[18] BERNARD CHAZELLE, *Computing on a free tree via complexity-preserving mappings*, Algorithmica, 2 (1987), pp. 337–361.

[19] BERNARD CHAZELLE AND C. SESHADHRI, *Online geometric reconstruction*, J. ACM, 58 (2011). Article No. 14.

[20] IRIT DINUR AND KOBBI NISSIM, *Revealing information while preserving privacy*, in Proceedings of the

twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, PODS '03, New York, NY, USA, 2003, ACM, pp. 202–210.

[21] YEVGENIY DODIS, ODED GOLDREICH, ERIC LEHMAN, SOFYA RASKHODNIKOVA, DANA RON, AND ALEX SAMORODNITSKY, *Improved testing algorithms for monotonicity*, in Proceedings of RANDOM 1999, Dorit S. Hochbaum, Klaus Jansen, José D. P. Rolim, and Alistair Sinclair, eds., vol. 1671 of Lecture Notes in Comput. Sci., Springer, 1999, pp. 97–108.

[22] CYNTHIA DWORK, FRANK MCSHERRY, KOBBI NISSIM, AND ADAM SMITH, *Calibrating noise to sensitivity in private data analysis*, in Theory of Cryptography, Shai Halevi and Tal Rabin, eds., vol. 3876 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, pp. 265–284. 10.1007/11681878_14.

[23] F. ERGUN, S. KANNAN, S. R. KUMAR, R. RUBINFELD, AND M. VISWANATHAN, *Spot-checkers*, J. Comput. Syst. Sci., 60 (2000), pp. 717–751.

[24] ELDAR FISCHER, *On the strength of comparisons in property testing.*, Inf. Comput., 189 (2004), pp. 107–116.

[25] ELDAR FISCHER, ERIC LEHMAN, ILAN NEWMAN, SOFYA RASKHODNIKOVA, RONITT RUBINFELD, AND ALEX SAMORODNITSKY, *Monotonicity testing over general poset domains*, in Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, STOC '02, New York, NY, USA, 2002, ACM, pp. 474–483.

[26] ODED GOLDREICH, SHAFI GOLDWASSER, ERIC LEHMAN, DANA RON, AND ALEX SAMORODNITSKY, *Testing monotonicity.*, Combinatorica, 20 (2000), pp. 301–337.

[27] ODED GOLDREICH, SHAFI GOLDWASSER, AND DANA RON, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[28] PARIKSHIT GOPALAN, RYAN O'DONNELL, YI WU, AND DAVID ZUCKERMAN, *Fooling functions of halfspaces under product distributions*, in IEEE Conference on Computational Complexity, IEEE Computer Society, 2010, pp. 223–234.

[29] ANDREAS HAEBERLEN, BENJAMIN C. PIERCE, AND ARJUN NARAYAN, *Differential privacy under fire*, in Proceedings of the 20th USENIX conference on Security, SEC'11, Berkeley, CA, USA, 2011, USENIX Association, pp. 33–33.

[30] SHIRLEY HALEVY AND EYAL KUSHILEVITZ, *Testing monotonicity over graph products*, Random Struct. Algorithms, 33 (2008), pp. 44–67.

[31] MORITZ HARDT AND KUNAL TALWAR, *On the geometry of differential privacy*, in Proceedings of the 42nd ACM symposium on Theory of computing, STOC '10, New York, NY, USA, 2010, ACM, pp. 705–714.

[32] MADHAV JHA AND SOFYA RASKHODNIKOVA, *Testing and reconstruction of Lipschitz functions with applications to data privacy*, in FOCS, Rafail Ostrovsky, ed., IEEE, 2011, pp. 433–442.

[33] SATYEN KALE, YUVAL PERES, AND C. SESHADHRI, *Noise tolerance of expanders and sublinear expander reconstruction*, in FOCS, IEEE Computer Society, 2008, pp. 719–728.

[34] BALA KALYANASUNDARAM AND GEORG SCHNITGER, *The probabilistic communication complexity of set intersection*, SIAM J. Discrete Math., 5 (1992), pp. 545–557.

[35] COLIN MCDIARMID, *On the method of bounded differences*, in Surveys in Combinatorics, Cambridge, 1989, Cambridge University Press, pp. 148–188.

[36] FRANK MCSHERRY, *Privacy integrated queries: an extensible platform for privacy-preserving data analysis*, Commun. ACM, 53 (2010), pp. 89–97.

[37] PRASHANTH MOHAN, ABHRADEEP THAKURTA, ELAINE SHI, DAWN SONG, AND DAVID CULLER, *GUPT: privacy preserving data analysis made easy*, in Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD '12, New York, NY, USA, 2012, ACM, pp. 349–360.

[38] KOBBI NISSIM, SOFYA RASKHODNIKOVA, AND ADAM SMITH, *Smooth sensitivity and sampling in private data analysis*, in STOC, David S. Johnson and Uriel Feige, eds., ACM, 2007, pp. 75–84.

[39] MICHAL PARNAS, DANA RON, AND RONITT RUBINFELD, *Tolerant property testing and distance approximation*, J. Comput. Syst. Sci., 72 (2006), pp. 1012–1042.

[40] SOFYA RASKHODNIKOVA, *Transitive-closure spanners: A survey*, in Property Testing, Oded Goldreich, ed., vol. 6390 of Lecture Notes in Computer Science, Springer, 2010, pp. 167–196.

[41] ALEXANDER A. RAZBOROV, *On the distributional complexity of disjointness*, Theor. Comput. Sci., 106 (1992), pp. 385–390.

[42] DANA RON, *Algorithmic and analysis techniques in property testing*, Foundations and Trends in Theoretical Computer Science, 5 (2009), pp. 73–205.

[43] INDRAJIT ROY, SRINATH T. V. SETTY, ANN KILZER, VITALY SHMATIKOV, AND EMMETT WITCHEL, *Airavat: security and privacy for mapreduce*, in Proceedings of the 7th USENIX conference on Networked systems design and implementation, NSDI'10, Berkeley, CA, USA, 2010, USENIX Association, pp. 20–20.

[44] RONITT RUBINFELD AND ASAF SHAPIRA, *Sublinear time algorithms*, SIAM J. Discrete Math., 25 (2011), pp. 1562–1588.

[45] RONITT RUBINFELD AND MADHU SUDAN, *Robust characterization of polynomials with applications to*

program testing, SIAM J. Comput., 25 (1996), pp. 252–271.

[46] MICHAEL E. SAKS AND C. SESHADHRI, *Local monotonicity reconstruction*, SIAM J. Comput., 39 (2010), pp. 2897–2926.

[47] MIKKEL THORUP, *Parallel shortcutting of rooted trees*, J. Algorithms, 23 (1997), pp. 139–159.

[48] ANDREW C. YAO, *Space-time tradeoff for answering range queries (extended abstract)*, in Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, eds., STOC '82, New York, NY, USA, 1982, ACM, pp. 128–136.