PROPERTY TESTING AND RECONSTRUCTION WITH APPLICATIONS TO

DATA PRIVACY

A Dissertation in
Computer Science and Engineering
by
Madhav Jha

© 2013 Madhav Jha

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2013
The dissertation of Madhav Jha was reviewed and approved* by the following:

Sofya Raskhodnikova  
Associate Professor of Computer Science and Engineering  
Dissertation Advisor, Chair of Committee

Piotr Berman  
Associate Professor of Computer Science and Engineering

Martin Fürer  
Professor of Computer Science and Engineering

Jason Morton  
Assistant Professor of Mathematics and Statistics

Raj Acharya  
Professor of Computer Science and Engineering  
Head of Department

*Signatures are on file in the Graduate School.
Abstract

The Lipschitz property is a fundamental property of functions with many applications in mathematics and computer science. Intuitively, a function is Lipschitz if it is not too sensitive to small changes in its inputs. In various applications, it is important (often crucial) that the input function satisfies the Lipschitz property. Given query access to a function, can we test that it is Lipschitz? Better still, can we restore or reconstruct the Lipschitz property in a (possibly non-Lipschitz) function by modifying it suitably? In this thesis, we initiate the study of testing and reconstruction of the Lipschitz property of functions. Our primary motivation for studying the Lipschitz property stems from its applications to data privacy.

Formally, a function $f : D \rightarrow R$ 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 investigate the Lipschitz property of functions under the well-studied model of property testing (Rubinfeld and Sudan; Goldreich, Goldwasser and Ron) and local property reconstruction (Ailon, Chazelle, Comandur and Liu; Saks and Seshadhri). 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 reconstructor restores 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 (or at least “close”) to $f$.

We design efficient testers and local reconstructors for functions over domains of the form $\{1, \ldots, n\}^d$, equipped with $\ell_1$ distance, and give corresponding impossibility results. The algorithms we design have applications to data privacy and program analysis.
# Table of Contents

## List of Figures

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>vii</td>
</tr>
</tbody>
</table>

## Acknowledgments

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>viii</td>
</tr>
</tbody>
</table>

## Chapter 1

**Introduction**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Lipschitz functions over finite domains</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Property testing</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1</td>
<td>Our results</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Local property reconstruction</td>
<td>7</td>
</tr>
<tr>
<td>1.3.1</td>
<td>Our results</td>
<td>8</td>
</tr>
<tr>
<td>1.4</td>
<td>Applications</td>
<td>10</td>
</tr>
<tr>
<td>1.4.1</td>
<td>Filter mechanism for data privacy</td>
<td>10</td>
</tr>
<tr>
<td>1.4.2</td>
<td>Testing privacy</td>
<td>13</td>
</tr>
<tr>
<td>1.4.3</td>
<td>Testing robustness of programs</td>
<td>14</td>
</tr>
<tr>
<td>1.5</td>
<td>Organization of the thesis</td>
<td>15</td>
</tr>
</tbody>
</table>

## Chapter 2

**Preliminaries**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Property testing</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Local property reconstruction</td>
<td>17</td>
</tr>
<tr>
<td>2.3</td>
<td>Facts about Lipschitz functions</td>
<td>18</td>
</tr>
</tbody>
</table>

## Chapter 3

**Testing if a function on $\mathcal{H}_d$ is Lipschitz**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>A Lipschitz tester for functions on the hypercube</td>
<td>20</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Averaging operator $A_i$</td>
<td>23</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A property reconstructor: g always satisfies property P.</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>Use of a Lipschitz filter in private data analysis.</td>
<td>10</td>
</tr>
<tr>
<td>5.1</td>
<td>The discrete derivative function, $\Delta f_i$.</td>
<td>47</td>
</tr>
<tr>
<td>6.1</td>
<td>Two cases in the proof of Claim 6.1.</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>Application of the basic operator on a square.</td>
<td>66</td>
</tr>
<tr>
<td>8.1</td>
<td>Functions used in proof of Lemma 8.2.</td>
<td>90</td>
</tr>
<tr>
<td>8.2</td>
<td>Functions used in proof of Lemma 8.5 for $a = 0$. Observe $f_0^0(x) = 1$ and $f_1^1(y) = 0$.</td>
<td>93</td>
</tr>
</tbody>
</table>
Acknowledgments

I would like to express my deepest gratitude to my advisor, Sofya Raskhodnikova. She has been the driving force behind pretty much everything that is part of this thesis. I would also like to thank my teachers and mentors (be it the faculty here at Penn State, or faculty at my undergraduate college, or my internship mentors, or my college seniors, or my school teachers) who have all shaped my thinking and pointed me in the right direction at various stages of my life. My collaborators in research have made my research journey ever so enjoyable, so special thanks go to all of them: Pranjal Awasthi, Arnab Bhattacharyya, Eric Blais, Kashyap Dixit, Elena Grigorescu, Kyomin Jung, Marco Molinaro, Ali Pinar, Sofya Raskhodnikova, C. Seshadhri, Abhradeep Guha Thakurta, David P. Woodruff, and Grigory Yaroslavtsev. Finally, I would like to thank my family and friends who have always been very supportive of all my endeavors, research or otherwise.

I also gratefully acknowledge the National Science Foundation whose grants (specifically, NSF/CCF CAREER award 0845701, NSF/CCF 0729171 and NSF grant CDI-0941553) supported my work that went into this thesis.
Dedication

Dedicated to my parents (Mrs. Sangeeta Jha and Mr. Mithilesh Jha).
Chapter 1

Introduction

The study of properties of functions is central to mathematics and computer science. In this thesis, we study Lipschitz continuity, a fundamental property of functions which finds many applications in these areas. Intuitively, a function is Lipschitz continuous if the change (increase or decrease) in the output of the function resulting from modifying one of its inputs arbitrarily is always bounded by a constant; the constant itself is called a Lipschitz constant of the function. As an example, consider the function which returns the sum of its inputs, where each input lies between 0 and $c$. Then changing one of the inputs arbitrarily can change the sum by at most $c$. Therefore, $c$ is a Lipschitz constant of this sum function. We say a function is $c$-Lipschitz if it has Lipschitz constant $c$.

Applications of a Lipschitz constant of a function abound in computer science. For instance, in the analysis of randomized algorithms, a Lipschitz constant of a function $f$ is used to obtain tail bounds via McDiarmid’s inequality [McD89a]; in program analysis, it measures robustness of programs to noise [CGLN11]; in data privacy, it is used to scale noise added to the output of a computation on a private database in order to preserve privacy [DMNS06]. In fact, the last application serves as our primary motivation for studying Lipschitz continuity.

Next we describe the privacy application in more detail. The main challenge in private data analysis is to release functions $f(x)$ of a private database $x$ accurately and without compromising privacy of individuals whose information is stored in the database. A rigorous and well-established notion of privacy, called differential privacy [DMNS06], requires that the (distribution of the) output of the mechanism for releasing $f(x)$ should not change too much by the presence or absence of an individual record of the database.
One of the basic ways of releasing $f(x)$ while satisfying differential privacy is to add random noise proportional to a Lipschitz constant of the function [DMNS06]. (Thus, for releasing the $c$-Lipschitz sum function discussed in the beginning of the introduction, it is sufficient to add noise proportional to $c$.) However, for this privacy mechanism to work, one must know a Lipschitz constant of the function $f$. This is especially problematic when $f$ is given as a program (e.g., as a C++ program) supplied by an untrusted (possibly malicious) external client. The previous approach was to either restrict the program $f$ to only use a specific set of trusted “library” functions (such as sum, count, etc.) for computations on the private data (e.g., [McS10, RSK+10]); or, allow only well-typed programs (in a specific type-system) for which a Lipschitz constant can be deduced at compile time (e.g., [HPN11]). But what if we wish to allow arbitrary programs?

In this thesis, we initiate the study of the $c$-Lipschitz property in the framework of local property reconstruction [ACCL07, SS10] and show that one can allow arbitrary\textsuperscript{1} programs in the above setting. Specifically, we develop an algorithm called the $c$-Lipschitz filter which, roughly speaking, can be used to replace oracle access to an arbitrary input function $f$ with oracle access to a related $c$-Lipschitz function $g$. Moreover, if the input function $f$ is already $c$-Lipschitz, then with high probability, $g$ is identical to $f$. The privacy mechanism can use the function $g$ (with $c$ as its Lipschitz constant) instead of the program $f$ sent by the client. This modified mechanism (called filter mechanism) is always differentially private. Moreover, the accuracy requirement of the Lipschitz filter ensures that it is as accurate as the original privacy mechanism for honest clients.

Zooming out from the specific privacy setting discussed above, consider the more general scenario: an algorithm $A$ (again originating from an untrusted client) computes on a private data set $x$ and purportedly preserves some notion of privacy. Can we test if $A$ indeed preserves the said notion of privacy? We show that for a variant of differential privacy, this question is intimately connected to knowing whether an input function satisfies the $c$-Lipschitz property. In this thesis, we initiate the study of testing if a function (over a finite domain) is $c$-Lipschitz. Specifically, we investigate the problem in the framework of property testing [GGR98, RS96] which is a well-studied notion of approximation for decision problems. The $c$-Lipschitz testers yield algorithms for

\textsuperscript{1}The only assumptions we make is that the program computing $f$ terminates and that the domain and range of $f$ are finite.
testing the variant of differential privacy mentioned above. Moreover, these testers are also applicable in program analysis.

The rest of the introduction is organized as follows. In Section 1.1, we define the Lipschitz property formally and describe functions that we consider in this thesis. Next we describe our results on testing and reconstruction in Sections 1.2 and 1.3, respectively. Applications of our algorithms are described in Section 1.4. We organize the thesis similarly to the introduction. The final section of the introduction briefly describes the thesis outline.

1.1 Lipschitz functions over finite domains

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

In various applications, given a function \( f \), it is often required to compute a Lipschitz constant of \( f \). Or, at least, verify that the input function \( f \) is \( c \)-Lipschitz for a given number \( c \). However, in general, computing a Lipschitz constant is 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 thesis, we focus on Lipschitz continuity of functions over finite domains, for which the NP-hardness statement still holds.

We study Lipschitz functions over discrete metric spaces. Throughout the thesis, 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 \(\{0,1\}^d, [n]^d\), equipped with the \(\ell_1\) distance. We identify points in the domain with vertices of a graph. The edges in the graph connect points which are at distance 1. We refer to the domains of our functions by specifying the resulting graph that captures the distances

---

2More 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 Chapter 5, 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 \(d_D(u,v) = \infty\).
between points in the domain. Specifically, \( \{0, 1\}^d \) is referred to as the hypercube \( \mathcal{H}_d \), \([n]\) as the line \( \mathcal{L}_n \) and \([n]^d\) as the hypergrid \( \mathcal{H}_{n,d} \). The hypergrid \( \mathcal{H}_{n,d} \) has vertex set \([n]^d\) and edge set \( \{\{x, y\} : \exists \text{ unique } i \in [d] \text{ such that } |y_i - x_i| = 1 \text{ and for } j \neq i, y_j = x_j\} \).

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

### 1.2 Property testing

Property testing is concerned with approximate decision making. 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 [Ron09, RS11] for recent surveys. A property tester is an algorithm which can distinguish between functions which satisfy a given property \( P \) (in our case, the Lipschitz property) from those which are \( \epsilon \)-far from the property, where \( \epsilon \in (0, 1) \) is a parameter given as input to the tester. A function \( f \) is \( \epsilon \)-far from property \( P \) if the distance between \( f \) and every member of \( P \) (where we view \( P \) as the set of functions satisfying \( P \)) is at least \( \epsilon \) under a suitable definition of distance between functions. In the standard property testing, the distance between functions \( f \) and \( g \) is the fraction of points of the domain on which \( f \) and \( g \) differ. Equivalently, the distance between \( f \) and \( g \) is given by \( \Pr[f(x) \neq g(x)] \) where \( x \) is chosen uniformly from the domain. In this case, we say that the underlying distribution on the domain is uniform. We also consider a more general setting where the underlying distribution on the domain is unknown. Property testing under unknown distribution has been studied under the name of distribution free testing [HK07] beginning with [GGR98]. However, most results in property testing are concerned with testing with respect to the uniform distribution. Throughout this thesis, if we do not qualify a tester, we mean a tester over the uniform distribution.

The notion of “distinguish” in the above description is formalized by requiring the tester to accept, with high probability \(^3\), inputs which satisfy the property. Moreover, the tester must reject, again with high probability, inputs which are \( \epsilon \)-far from the property. These testers are called 2-sided error testers to differentiate them from 1-sided error testers, where the latter testers never reject inputs which satisfy the property. We also differentiate between adaptive and non-adaptive testers. While an adaptive tester can

---

\(^3\)By high probability, we mean with probability at least 2/3.
query its input based on answers to previous queries, a non-adaptive tester must make all its queries in advance. Observe that this differentiation (between adaptive and non-adaptive) can be made for any algorithm which queries it’s input.

1.2.1 Our results

In this section, we describe our results on testing the Lipschitz property. We design efficient testers of the Lipschitz property for functions over the hypercube $\mathcal{H}_{d}$, the line $\mathcal{L}_{n}$ and the hypergrids $\mathcal{H}_{n,d}$. The performance of our testers is better when the image space of the function has low diameter. Therefore, we express the running time of our results in terms of the image diameter of function $f$. The image diameter of function $f$, denoted $\text{ImD}(f)$, is the difference between the maximum and the minimum values attained by $f$. We present results for real-valued functions as well as functions whose range is $\delta \mathbb{Z}$, namely, the set of integer multiples of $\delta$ for $\delta \in (0, 1]$. All our algorithms are non-adaptive.

- **Testing the Lipschitz property on $\mathcal{H}_{d}$.** We design a 1-sided error Lipschitz tester for functions $f : \mathcal{H}_{d} \to \delta \mathbb{Z}$ which runs in time $O\left(\frac{d \min(d, \text{ImD}(f))}{\delta \epsilon}\right)$. This tester implies an algorithm, with (asymptotically) the same running time, for real-valued functions on $\mathcal{H}_{d}$, with guarantees similar to a Lipschitz tester: the algorithm can distinguish Lipschitz functions from those which are $\epsilon$-far from $(1 + \delta)$-Lipschitz with high probability. We also prove that any (possibly adaptive and 2-sided error) tester for the Lipschitz property of functions on $\mathcal{H}_{d}$ must make $\Omega(d)$ queries even for functions with range $\{0, 1, 2\}$. These results are described in Chapter 3.

- **Testing the Lipschitz property on $\mathcal{H}_{d}$ over a product distribution.** We show how to test the Lipschitz property on $\mathcal{H}_{d}$ even when the underlying distribution on the domain is an unknown product distribution. A distribution on $\{0, 1\}^{d}$ is called a product distribution if its marginal distributions are mutually independent. The testers have similar guarantees as the testers for the uniform distribution except for the running time which has a slightly worse dependence on $\delta$. The tester and its analysis are presented in Chapter 4.

---

4As is standard in property testing, the running time does not include the time to generate random samples. In other words, it is assumed that the query to the input can be made in constant time.
• **Testing edge transitive properties that allow extension.** In Chapter 5, we show that the Lipschitz property of functions $f : \mathcal{L}_n \to \mathbb{R}$ can be tested in $O(\log n/\epsilon)$ time for a large class of metric spaces $(R, d_R)$, which includes $(\mathbb{R}^k, \ell_p)$ 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 $G = (V, E)$. In fact, we obtain this result as a corollary of a general result which gives a property tester for a large class of properties of functions, namely, for every *edge-transitive* property that allows extension. We formally define this class in Chapter 5. Another important property that falls in this class is the *monotonicity* property of functions $f : G_n \to \mathbb{R}$ where $G_n$ is a directed acyclic graph (DAG). A function $f : G_n \to \mathbb{R}$ defined on the vertex set of a DAG $G_n$ is *monotone* if $f(x) \leq f(y)$ holds for every edge $(x, y)$ in $G_n$. We show that the monotonicity tester from [BGJ+12b] is, in fact, a tester for this general class of properties. Monotonicity testing is a very well studied property of functions [EKK+00, GGL+00, DGL+99, BRW05, FLN+02, Fis04, HK08, AC06, BGJ+12b, BGJ+12a, BCGSM12, PRR06, ACCL07, CS13a, CS13b]. Several techniques from some of these works have proved surprisingly useful for testing the Lipschitz property. Returning again to the Lipschitz property of functions on the line, we show that, for real-valued functions, the running time mentioned above can be improved to $O\left(\min\{\log n, \text{ImD}(f)\}/\epsilon\right)$. We also give a matching lower bound for non-adaptive 1-sided error testers for these functions.

• **Testing the Lipschitz property on $\mathcal{H}_{n,d}$.** In Chapter 6, we show that the Lipschitz property of functions $f : \mathcal{H}_{n,d} \to \delta\mathbb{Z}$ can be tested nonadaptively and with 1-sided error in

$$O\left(\frac{d}{\delta \epsilon} \cdot \min \{\text{ImD}(f), nd\} \cdot \log \min \{\text{ImD}(f), n\}\right)$$

time. Allowing for 2-sided error allows us to improve the running time slightly. Specifically, the $d$ inside the min function can be replaced with $\sqrt{d \log(1/\epsilon)}$.

The main tool in the analysis of our tester is a smoothing procedure that makes a function Lipschitz by modifying it at a few points. Its analysis is already non-trivial for the 1-dimensional version, which we call Bubble Smooth, in analogy to Bubble Sort. In one step, Bubble Smooth modifies two values that violate the Lipschitz property, namely, differ by more than 1, by transferring $\delta$ units from the
larger to the smaller. We define a *transfer graph* to keep track of the transfers, and use it to show that the $\ell_1$ distance between $f$ and BubbleSmooth($f$) is at most twice the $\ell_1$ distance from $f$ to the nearest Lipschitz function. Bubble Smooth has several other important properties, which allow us to obtain a *dimension reduction*, i.e., a reduction from testing functions on multidimensional domains to testing functions on the 1-dimensional domain, that incurs only a small multiplicative overhead in the running time and thus avoids the exponential dependence on the dimension.

Subsequent to our work, Chakrabarty and Seshadhri [CS13b] gave a Lipschitz tester for functions $f : \mathcal{H}_{n,d} \rightarrow \mathbb{R}$ with respect to the uniform distribution which runs in time $O((d \log n) / \epsilon)$. This improves our results on testing the Lipschitz property of functions over $\mathcal{H}_{n,d}$ (for $d > 1$) and over $\mathcal{H}_d$ (for functions with non constant image diameter).

### 1.3 Local property reconstruction

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 [ACCL08] is a direction of research in sublinear algorithms that has its roots in property testing. Some related notions include locally decodable codes [KT00], program checking [BLR93] and, more generally, local computation [RTVX11, ARVX12]. Property-preserving data reconstruction 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 [SS10] 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 2.5), defined in [BGJ+12a], further requires that if the original input has the property, it should not be modified by the filter, i.e., if $f$ has the property, $g$ must be equal to $f$. Our application to data privacy has an unusual feature, not encountered in previous applications of filters: algorithm $A$ needs to access its input only at one point $x$ (corresponding to the database its holding). Nevertheless, we require local filters, not because of the distributed aspect they were initially developed for, but because when $g$ depends on $x$, it might leak information about $x$ and violate privacy.

1.3.1 Our results

- A deterministic local filter of the Lipschitz property. In Chapter 7, we give a deterministic local filter of the Lipschitz property for functions of the form $f : [n]^d \rightarrow \mathbb{R}$ which runs in time $O((\log n)^d)$. To obtain this result, we abstract the combinatorial object used in 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 7.2.) 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

---

**Figure 1.1.** A property reconstructor: $g$ always satisfies property P.
in $H$ is the size of the set of vertices reachable from the node in $H$. We obtain a lookup graph for $[n]$ with maximum reachable-degree (of every node in $H$) bounded by $O(\log n)$. Our construction builds on ideas of Ailon et al. [ACCL08] who gave a local monotonicity filter for functions $f : [n] \to \mathbb{R}$. We obtain a lookup graph for the hypergrid $H_{n,d}$ by constructing a strong product of the lookup graphs for the line.

- **Limitations of local filter for Lipschitz and monotone functions.** Our Lipschitz filter runs in time sublinear in the size of the domain of $f$. However, the running time of the filter has exponential dependence in the dimension $d$. This motivates the question about limits of property reconstruction. In Chapter 8, we study limitations of local filters for two properties of functions: the Lipschitz property and the monotonicity property. The first property is motivated by the privacy application and the second is a ‘benchmark’ problem in property-preserving reconstruction and property testing.

In trying to understand limitations of local reconstruction, we relax the notion of local filter (as described in Section 1.3) further. Specifically, we allow the reconstructed function $g$ to differ from $f$ by a small amount on every point, even if $f$ satisfies the property: namely, we require that with high probability, for every $x$ in the domain, $|g(x) - f(x)| \leq a$ for some fixed parameter $a$. Local filters as defined previously are a special case of this definition with $a = 0$. Therefore, a lower bound on this relaxed definition of filters, which we call $\alpha$-filter, implies a lower bound on filters considered in Section 1.3. We remark that local filters with small additive error can still be used in the privacy application described in the introduction. (Details appear in Chapter 8.)

We show that every local filter for Lipschitz or monotone functions runs in time exponential in the dimension $d$, even when the filter is allowed significant additive error. To prove our lower bounds, we construct families of hard functions and show that lookups of a local filter on these functions are captured by a combinatorial object that we call a $c$-connector. Then we present a lower bound on the maximum outdegree of a $c$-connector, and show that it implies the desired bounds on the running time of local filters. Our lower bounds, in particular, imply the same bound on the running time for a class of privacy mechanisms based on
filters (see Section 1.4.1 below).

## 1.4 Applications

Our Lipschitz filters and testers have applications to data privacy. We briefly describe these applications in Sections 1.4.1 and 1.4.2 below. Full details appear in Chapters 9 and 10. In addition, Lipschitz testers also find application in program analysis. This is described in Section 1.4.3.

![Figure 1.2. Use of a Lipschitz filter in private data analysis.](image)

### 1.4.1 Filter mechanism for 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 [DN03] pioneering a line of work in cryptography. Dwork et al. [DMNS06] defined a rigorous notion of privacy, called *differential privacy* (reviewed in Defition 9.1), 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 9.2. 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\(^5\): PINQ [McS10], Airavat [RSK+10] and Fuzz [HPN11]. All allow releasing functions of the database of the form $f : x \to \mathbb{R}$. In all implementations, the client sends a program to the server, requesting to evaluate it on the database, and receives the output of the program with Laplace noise added to it. However, the client is not trusted to provide a function with a low Lipschitz constant. The 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 Chapter 9, 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

\(^5\)Subsequent to our work, another system, GUPT [MTS+12], appeared. Instead of the Laplace mechanism, it is based on Sample and Aggregate [NRS07]. 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, 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\(^6\). 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.\(^7\)

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.

\(^6\)If one needs to ensure that a function is \(c\)-Lipschitz, the function can be rescaled.

\(^7\) The definition of local filters we use (Definition 2.5), unlike the original one proposed by Saks and Seshadhri [SS10], 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 \(\max_y \{|f(y) - f(x)| + d_G(x, y)\}\).
1.4.2 Testing privacy

Our motivation for enforcing the Lipschitz property on the input program \( f \), described in Section 1.4.1, was to ensure that the overall privacy mechanism which uses the reconstructed version of \( f \) satisfies differential privacy. Thus, our goal was to ensure that an algorithm \( \mathcal{A} \) computing on a sensitive database meets a specific privacy-preserving specification. Can we directly attack this goal? Namely, given an algorithm (e.g., as a program), can we test (or better still enforce) that it is private? The problem is especially challenging because the algorithm (just like the program \( f \) in Section 1.4.1) might originate from untrusted or malicious clients.

We formulate the above problem in the spirit of property testing and introduce the notion of testing privacy\(^8\). Specifically, we develop randomized algorithms which can distinguish, with high probability, whether an input “algorithm” \( \mathcal{A} \) satisfies differential privacy, or, fails to satisfy even a relaxation of differential privacy. The specific relaxation of differential privacy that we adhere to is a distributional relaxation of differential privacy. We call our relaxation differential privacy on typical datasets (DPTD). The privacy guarantees provided by DPTD are similar to differential privacy except they now hold only for typical datasets, namely, for datasets which are more likely to occur assuming a fixed (but unknown) probability distribution on the set of all datasets. We remark that DPTD is a special case of generalized differential privacy from [BBG+11a].

To develop algorithms described above, we must decide on the nature of access that we have to the input algorithm \( \mathcal{A} \). In this work, we make a few simplifying assumptions regarding access to \( \mathcal{A} \). We take the domain of \( \mathcal{A} \) to be the set of databases where each database is a fixed length vector over a finite alphabet. Moreover, we assume that the range of \( \mathcal{A} \) is finite. Finally, we assume constant-time access to the value \( \Pr[\mathcal{A}(x) = r] \) for every \( x \) and \( r \) in \( \mathcal{A} \)’s domain and range, respectively. (Recall that \( \mathcal{A} \) is a randomized algorithm. Indeed, nontrivial differentially private algorithms must be randomized.)

With these assumptions, we can identify \( \mathcal{A} \) with a finite family of functions, where we have query access to each function in the family. (Specifically, the family consists of functions \( f_r \) for every \( r \) in the range of \( \mathcal{A} \), where each \( f_r \) has the same domain as \( \mathcal{A} \) and is defined as \( f_r(x) = \Pr[\mathcal{A}(x) = r] \).) We show that our problem described above (which we refer to as testing DPTD) reduces to testing (in the sense of property testing)

\(^8\)We use the term “testing privacy” to describe the problem formulated here. This should not be confused with the property testing framework described in the introduction.
that every function in the family is simultaneously Lipschitz.

DPTD is defined with respect to a distribution on the set of datasets. The distribution could be not only arbitrary but even unknown. Therefore, in order to test DPTD with respect to an arbitrary and unknown distribution, we ideally need distribution-free Lipschitz testers. However, the evidence about testing a related property (the monotonicity property) suggests that efficient testers may not exist. Nevertheless, as described in Section 1.2.1, we developed an efficient Lipschitz tester which works for an arbitrary unknown product distribution over \( \{0, 1\}^d \). This gives a DPTD tester with respect to the same class of distributions.

Going beyond privacy testing, we also show how to convert an arbitrary algorithm \( \mathcal{A} \) into an algorithm \( \mathcal{A}' \) which always satisfies DPTD. Moreover, if \( \mathcal{A} \) is differentially private, then \( \mathcal{A}' \) has the same output distribution as \( \mathcal{A} \). This can be seen as a “filter” for privacy. The filter can be based on any DPTD tester. The results of this section are described in Chapter 10.

### 1.4.3 Testing robustness of programs

Certifying that a program computes a Lipschitz function has been studied in [CGLN11]. 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 [CGLN11] 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 [CGLN11].) 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.

1.5 Organization of the thesis

As mentioned earlier, we organized the introduction similarly to the thesis itself. In particular, Section 1.2 gave an overview of Chapters 3, 4, 5, and 6. Chapters 7 and 8 were described in Section 1.3. Finally, Sections 1.4.1 and 1.4.2 of the introduction were devoted to Chapters 9 and 10, respectively. To avoid repetition, we keep this outline section brief.

Outline of the thesis. Chapter 2 is devoted to preliminaries. Chapters 3, 5, and 6 describe our results on testing the Lipschitz property over the uniform distribution for functions on $\mathcal{H}_d$, $\mathcal{L}_n$, and $\mathcal{H}_{n,d}$, respectively. In addition, Chapter 5 also describes testing results for edge-transitive properties which allow extension. Chapter 4 studies testing the Lipschitz property over unknown product distribution. It shows that the hypercube tester of Chapter 3 can be extended to work even when the distribution on vertices of $\mathcal{H}_d$ is unknown, as long as the distribution is a product distribution. Chapter 7 gives a deterministic local Lipschitz filter for functions on $\mathcal{H}_{n,d}$. We explore limitations of local filters for Lipschitz and monotone functions in Chapter 8. Chapters 9 and 10 are devoted to applications of our algorithms to data privacy. Specifically, in Chapter 9, we introduce the filter mechanism for data privacy as described in Section 1.4.1. We also instantiate the filter mechanism with local filter of Chapter 7. Chapter 10 describes our results on testing privacy. Finally, we conclude with possible directions for future work in Chapter 11.
Chapter 2

Preliminaries

2.1 Property testing

Property testing is concerned with distinguishing objects which satisfy a given property \( \mathcal{P} \) from those which are far from satisfying it. We focus on properties of functions over finite domains. Let \( \mathcal{U} \) denote the set of all functions on 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 \( f \in \mathcal{P} \). The distance of a function \( f : D \to R \) to a property \( \mathcal{P} \) is usually the fraction of points of \( D \) on which \( f \) must be modified in order to satisfy the property. We use a more general notion of distance, first formulated by [GGR98], which is defined with respect to a probability distribution on the domain \( D \).

**Definition 2.1** (Distance to a property). Let \( \mathcal{P} \) be a property (i.e., a set) of functions \( f : D \to R \). Let \( \Pi \) be a distribution on \( D \). The distance \( \text{dist}_\Pi(f, g) \) between functions \( f, g : D \to R \) (with respect to the distribution \( \Pi \)) is \( \Pr_{x \sim \Pi}[f(x) \neq g(x)] \). The distance \( \text{dist}_\Pi(f, \mathcal{P}) \) of a function \( f \) to the property \( \mathcal{P} \) is \( \min_{g \in \mathcal{P}} \text{dist}_\Pi(f, g) \). For convenience, we use \( \epsilon_f \) as a shorthand for \( \text{dist}_\Pi(f, \mathcal{P}) \). We say that \( f \) is \( \epsilon \)-far from property \( \mathcal{P} \) if \( \epsilon_f \geq \epsilon \).

**Definition 2.2** (Property tester). A property tester is a randomized algorithm for testing property \( \mathcal{P} \) of functions \( f : D \to R \) with respect to distribution \( \Pi \) on \( D \), that gets oracle access to function \( f : D \to R \), oracle access to independent samples from distribution \( \Pi \) on \( D \), and a proximity parameter \( \epsilon \in (0, 1] \). The tester satisfies the following:
1. If \( f \) satisfies \( P \), then tester accepts with probability at least \( \frac{2}{3} \).

2. If \( f \) is \( \epsilon \)-far from property \( P \) with respect to distribution \( \Pi \), then tester rejects with probability at least \( \frac{2}{3} \).

A tester has one-sided error if it always accepts functions satisfying \( P \). A tester is nonadaptive if it makes all queries to its oracle \( f \) in advance, before receiving any answers from the oracle.

For the Lipschitz property, we also consider a relaxed version of property tester defined above. We note that this relaxation, given below in Definition 2.3, has been considered earlier in property testing, for example, in [PR02] for the property of having bounded diameter in graphs.

**Definition 2.3** ((1 + \( \delta \))-approximate Lipschitz tester). Consider the relaxation of definition of property tester, where Item 2 of Definition 2.2 is relaxed, so that, only if \( f \) is \( \epsilon \)-far from property \( P' \supset P \), the tester rejects with probability at least \( \frac{2}{3} \). (Item 1 of Definition 2.2 should still hold for \( P \).) When \( P \) is the Lipschitz property, and \( P' \) is the property of being \( (1 + \delta) \)-Lipschitz, we refer to this relaxed notion of property tester as \((1 + \delta)\)-approximate Lipschitz tester. In this context, we refer to \( \delta \) as the approximation parameter.

When we do not mention anything about distribution \( \Pi \) on the domain, we mean the uniform distribution. In such cases, we also omit \( \Pi \) from various notations involving \( \Pi \) mentioned above. Apart from the uniform distribution, we also consider product distributions \( \Pi \) on domain \([n]^d\).

**Definition 2.4** (Product distribution). A distribution \( \Pi \) on \([n]^d\) is called a product distribution if its marginal distributions are mutually independent. In other words, \( \Pi = \Pi_1 \times \Pi_2 \cdots \times \Pi_d \), where \( \Pi_i \) is a distribution on \([n]\).

### 2.2 Local property reconstruction

In this thesis, we consider local property reconstruction. The model of local property reconstruction was defined in [SS10], and the variant we consider was given in [BGJ+12a].
Definition 2.5 (Local filter). A local filter for reconstructing property $\mathcal{P}$ is an algorithm $A$ that has oracle access to a function $f : D \to \mathbb{R}$ and to an auxiliary random string $\rho$ (the “random seed”), and takes as input $x \in D$. For fixed $f$ and $\rho$, $A$ runs deterministically on input $x$ to produce an output $A_{f,\rho}(x) \in \mathbb{R}$. (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 $\rho$. 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 $\rho$.

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 [SS10], the authors also require that $g$ is sufficiently close to $f$: with high probability (over the choice of $\rho$), $\text{Dist}(g, f) \leq B(n) \cdot \text{Dist}(f, \mathcal{P})$, where $B(n)$ is a slowly growing function. We use the variant from [BGJ+12a] on this definition that does not have this requirement. (See Footnote 7.) We also consider a further relaxation of this definition in proving limitations of local filter. This relaxation is discussed in Chapter 8.

2.3 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 2.6. A function $f : D \to \mathbb{R}$ is Lipschitz on $D' \subseteq D$ if there are no violated pairs in $D' \times D'$.

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

Fact 2.1 (Lemma 1.1, [BL00]). 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 $D \setminus D'$.

Another fact about a Lipschitz function $f : D \to \mathbb{R}$ worth noting is that the largest distance between a pair of points in $f$’s range space (called $f$’s image diameter) cannot
exceed the largest distance between a pair of points in it’s domain space (called $D$’s 

diameter). We define the notion of diameter formally next.

**Definition 2.7** (Diameter). The diameter of a metric space $(D, d_D)$, denoted $\text{diam}(D)$, 
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 $\text{ImD}(f)$, is the difference between the 
maximum and the minimum values attained by $f$, i.e., $\max_{x \in D} f(x) - \min_{x \in D} f(x)$.

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 
$f : G \to \mathbb{R}$, that maps vertices of $G$ to $\mathbb{R}$, 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 $f : 
G \to \mathbb{R}$ 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 5.2 
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 2.8** (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 
[Ras10] for a survey on the topic) are used in Chapters 5 and 8.

**Definition 2.9** ($k$-TC-spanner, [BGJ$^+$12b]). 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 $d_H(x, y) \leq k$. 
Testing if a function on $\mathcal{H}_d$ is Lipschitz

In this chapter, we first show how to test functions defined on the hypercube for the Lipschitz property. Later, in Section 3.2, we prove a lower bound on the query complexity of testing the Lipschitz property on the hypercube. Recall that without any qualification to the testing problem, we always mean testing with respect to the uniform distribution. In particular, except for Lemma 3.2, which refers to testing with respect to arbitrary distribution, this entire chapter is devoted to testing the Lipschitz property with respect to the uniform distribution.

3.1 A Lipschitz tester for functions on the hypercube

The following theorem gives a tester for the Lipschitz property of functions of the form $f : \mathcal{H}_d \to \delta\mathbb{Z}$, where recall $\delta\mathbb{Z}$ is the set of integer multiples of $\delta$. Also, recall from Definition 2.7, that the image diameter of function $f$, denoted $\text{ImD}(f)$, is the difference between the maximum and the minimum values attained by $f$.

**Theorem 3.1** (Lipschitz tester for hypercube). The Lipschitz property of functions $f : \mathcal{H}_d \to \delta\mathbb{Z}$ can be tested nonadaptively and with one-sided error in $O\left(\frac{d \cdot \min\{d, \text{ImD}(f)\}}{\delta \epsilon}\right)$ time for all $\delta \in (0, 1]$.\(^1\)

---

\(^1\)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.
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 $O(d^2/\delta \epsilon)$. When designing a tester of the Lipschitz property of functions $f : \mathcal{H}_d \rightarrow \delta \mathbb{Z}$, 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 $g : \mathcal{H}_d \rightarrow \mathbb{Z}$ be the function $f/\delta$. 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\footnote{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 $f : S \rightarrow \mathbb{Z}$ be defined as follows: $f((0, 0)) = 0$, $f((0, 1)) = 1$ and $f((1, 1)) = 2$. Then, with respect to the $\ell_2$-metric on the domain, $f$ is $\sqrt{2}$-Lipschitz but not $\lfloor \sqrt{2} \rfloor$-Lipschitz.}. 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 $f : \mathcal{H}_d \rightarrow \delta \mathbb{Z}$ is Lipschitz is equivalent to testing if $f' : \mathcal{H}_d \rightarrow (1/c)\mathbb{Z}$ is Lipschitz for the integer $c$ defined above.

The main tool in the analysis of our test is the following lemma, proved in Section 3.1.1. Recall from the chapter on preliminaries (Chapter 2) that an edge $\{x, y\}$ of $\mathcal{H}_d$ is a violated edge, if $f$ fails to satisfy the Lipschitz property on $\{x, y\}$: $|f(x) - f(y)| > 1$.

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

Lemma 3.1 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. We state this lemma more generally than required in this section because the general statement is useful in subsequent chapters (specifically, in Chapters 4 and 5).

**Lemma 3.2 (Lipschitz tester oblivious to the image diameter).** Suppose there is a non-adaptive, 1-sided error tester for the Lipschitz property of functions $f : D \rightarrow \mathbb{R}$ with respect to distribution $\Pi$ on $D$. Assume further that this tester knows an upper bound $r$ on the image diameter $\text{ImD}(f)$ of the input function and runs in time $T(\epsilon, r)$. Then this
property can be tested nonadaptively and with one-sided error in time

\[ 2 \cdot T(\epsilon/2, \min \{\text{diam}(D), \text{Im}(f)\}) + O(1/\epsilon) \]

with no knowledge of an upper bound on \( \text{Im}(f) \).

**Proof of Theorem 3.1** In light of Lemma 3.2, 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.

The tester accepts all Lipschitz functions. Since the number of edges in the \( d \)-dimensional hypercube is \( 2^{d-1}d \), Lemma 3.1 implies that functions which are \( \epsilon \)-far from Lipschitz are rejected with probability at least 2/3. The theorem (including the claim about the running time) then follows from Lemma 3.2.

Next we obtain the following corollary for real-valued functions from Theorem 3.1 by discretizing their values.

**Corollary 3.1.** 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 \( \epsilon \)-far from \((1 + \delta')\)-Lipschitz and runs in \( O\left(\frac{d \min \{d, \text{Im}(f)\}}{\delta \epsilon} \right) \) time.

We start by defining the rounding operators used in the proof of the corollary and later in Section 3.1.1.

**Definition 3.1 (Operators \( \lfloor \cdot \rfloor_\delta \) and \( \lceil \cdot \rceil_\delta \)).** 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 3.1** 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 \( f(x) - \delta' \leq f'(x) \leq f(x) \) for all \( x \in \{0,1\}^d \). If \( f \) is Lipschitz then \( f' \) is \((1 + \delta')\)-Lipschitz because \(|f'(x) - f'(y)| \leq |f(x) - f(y)| + \delta' \leq 1 + \delta' \) for each edge \( \{x, y\} \) of the hypercube \( \mathcal{H}_d \). Next we show that when \( f \) is \( \epsilon \)-far from \((1 + 2\delta')\)-Lipschitz then \( f' \) is \( \epsilon \)-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 2.6 and Fact 2.1.) Since \(f'(x) \leq f(x) \leq f'(x) + \delta'\), function \(f\) is \((1 + 2\delta')\)-Lipschitz on \(S\), a contradiction.

Thus, we can use the Lipschitz tester of Theorem 3.1 with inputs \(f'/(1 + \delta')\), \(d\), \(\delta'/(1 + \delta')\) and \(\epsilon\) to distinguish Lipschitz \(f\) from \(f\) that is \(\epsilon\)-far from \((1 + \delta)\)-Lipschitz, proving the corollary.

\[\square\]

### 3.1.1 Averaging operator \(A_i\)

This section is devoted to Lemma 3.1, the main tool in the analysis of the tester of the Lipschitz property on the hypercube. To prove Lemma 3.1, we show how to transform an arbitrary function \(f : \{0, 1\}^d \rightarrow \delta\mathbb{Z}\) 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 \(H_d\) 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 \(\delta\) 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 3.2** (Basic operator \(B_i\)). Given \(f : \{0, 1\}^d \rightarrow \delta\mathbb{Z}\), 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 3.3** (Averaging operator \(A_i\)). Given \(f : \{0, 1\}^d \rightarrow \delta\mathbb{Z}\), 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 3.1. The new definition is equivalent to Definition 3.3, 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 3.4** (Averaging operator $A_i$, equivalent definition). Given $f : \{0, 1\}^d \rightarrow \delta \mathbb{Z}$, 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

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

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 3.5** (Violation score). The violation score of an edge $\{x, y\}$ with respect to function $f$, denoted $vs(\{x, y\})$, 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 $[\delta, \text{ImD}(f)]$. Let $V^i(f)$ be the number of edges along dimension $i$ violated by $f$.

$$\delta V^i(f) \leq VS^i(f) \leq V^i(f) \cdot \text{ImD}(f). \quad (3.1)$$

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 3.3.** For all $i, j \in [d]$, where $i \neq j$, and every function $f : \{0, 1\}^d \rightarrow \delta \mathbb{Z}$, applying the averaging operator $A_i$ does not increase the violation score in dimension $j$, i.e., $VS_j(A_i[f]) \leq VS_j(f)$.
**Proof** Given Definition 3.3, in which the averaging operator is viewed as multiple applications of the basic operator, it suffices to prove Lemma 3.3 for $B_i$ instead of $A_i$.

Note that the edges along dimensions $i$ and $j$ form disjoint squares in the hypercube. Therefore, the special case of Lemma 3.3 for $f$ restricted to each of these squares individually (where each such restriction is a two-dimensional function) allows us to prove the lemma for dimensions $i$ and $j$ by summing the inequalities over all such squares. It remains to prove the lemma for $d = 2$ and $B_i$ instead of $A_i$. In this proof, we use the fact that $1/\delta$ is integral, discussed in the beginning of Section 3.1.

Consider a two-dimensional function $f : \{x_t, x_b, y_t, y_b\} \rightarrow \delta\mathbb{Z}$ with vertices $x_t, x_b, y_t,$ and $y_b$ 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 $f(x_t)$ is symmetrical). Then $\{x_t, y_t\}$ is violated with $f(x_t) < f(y_t)$. Moreover, since $f$ is a $\delta\mathbb{Z}$-valued function and $1/\delta$ is an integer, $f(y_t) \geq f(x_t) + 1 + \delta$. The application of the basic operator increases $f(x_t)$ by $\delta$ and decreases $f(y_t)$ by $\delta$.

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

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

**Proof of Lemma 3.1** The crux of the proof is showing how to make a function \( f : \{0, 1\}^d \rightarrow \delta \mathbb{Z} \) Lipschitz by redefining it on at most \( 2\delta \cdot V(f) \cdot \text{ImD}(f) \) 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,

\[
\begin{align*}
    f &= f_0 \xrightarrow{A_1} f_1 \xrightarrow{A_2} f_2 \rightarrow \cdots \rightarrow f_{d-1} \xrightarrow{A_d} f_d.
\end{align*}
\]

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 3.3, \( A_i \) preserves the Lipschitz property along dimensions fixed in the previous steps. Thus, eventually there are no violated edges, and \( f_d \) is Lipschitz.

Now we bound the number of points on which \( f \) and \( f_d \) differ. For convenience, let \( \text{Dist}(f, g) \) denote the absolute distance between functions \( f \) and \( g \), i.e., \( \text{Dist}(f, g) = 2^d \cdot \text{dist}(f, g) \). For all \( i \in [d] \), we have,

\[
\begin{align*}
    \text{Dist}(f_{i-1}, f_i) &= \text{Dist}(f_{i-1}, A_i[f_{i-1}]) \\
    &\leq 2 \cdot V^i(f_{i-1}) \leq \frac{2}{\delta} \cdot V S^i(f_{i-1}) \leq \frac{2}{\delta} \cdot V S^i(f) \\
    &\leq \frac{2}{\delta} \cdot V^i(f) \cdot \text{ImD}(f). \quad (3.2)
\end{align*}
\]

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 3.3, 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

\[
\begin{align*}
    \text{Dist}(f, f_d) &\leq \sum_{i \in [d]} \text{Dist}(f_{i-1}, f_i) \\
    &\leq \sum_{i \in [d]} \frac{2}{\delta} \cdot V^i(f) \cdot \text{ImD}(f) \quad (3.3) \\
    &= \frac{2}{\delta} \cdot V(f) \cdot \text{ImD}(f). \quad (3.4)
\end{align*}
\]

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

Consider a function \( f \) which is \( \epsilon \)-far from the Lipschitz property. Since \( f_d \) is Lips-
chitz, \( \text{Dist}(f, f_d) \geq \epsilon \cdot 2^d \). Together with (4.2), it gives \( V(f) \geq \epsilon \delta \cdot 2^{d-1} / \text{ImD}(f) \), as required.

3.1.2 Analyzing the image diameter of a sample

In this section, we prove Lemma 3.2 which is used in the proofs of Theorems 3.1, 4.1 and 5.2. 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 3.2.

**Claim 3.1.** Consider a function \( f : D \rightarrow \mathbb{R} \) and let \( \Pi \) be a distribution on \( D \). Given \( \epsilon \in (0, 1] \), let \( z = z_1, \ldots, z_s \) be independent and identically distributed samples from \( \Pi \) of size \( s = \lceil 6/\epsilon \rceil \). Let \( r = \max_{i=1}^s f(z_i) - \min_{i=1}^s f(z_i) \) denote the image diameter of \( f \) on the sample. Then, value \( r \) is at most \( \text{ImD}(f) \). Moreover, with probability at least \( 5/6 \), function \( f \) is \( \epsilon \)-close to having image diameter \( r \), where the distance to the property of “having image diameter at most \( r \)” is with respect to distribution \( \Pi \). (See Definition 2.1.)

**Proof.** In this proof, all probabilities are with respect to distribution \( \Pi \) on \( D \). Also, given a finite set \( S \), we use notation \( \Pr(S) \) to denote the sum of probability mass of elements in the set \( S \). The claim that \( r \) is at most \( \text{ImD}(f) \) follows immediately from the definition of \( r \). Next we prove the second claim about \( r \). Let \( x_1, \ldots, x_{2^d} \) be points in \( \{0, 1\}^d \) sorted according to their function values in nondecreasing order. Let \( L_i \) (respectively, \( R_i \)) denote the first (respectively, the last) \( i \) points in the ordering. Let \( u \) be the smallest index \( i \) such that \( \Pr(L_i) \geq \epsilon/2 \). Similarly, let \( v \) be the largest index \( i \) such that \( \Pr(R_i) \geq \epsilon/2 \). Definition of \( u \) implies that probability mass of set \( L_{u-1} \) is at most \( \epsilon/2 \). Because otherwise we would have \( \Pr(L_{u-1}) \geq \epsilon/2 \) contradicting the minimality of \( u \). Analogously, \( \Pr(R_{v+1}) \leq \epsilon/2 \).

Let \( E_1 \) (respectively, \( E_2 \)) denote the event that the sampled sequence \( z \) contains no element of \( L_{u-1} \) (respectively, \( R_{v+1} \)). Observe that if \( z \) contains an element of \( L_{u-1} \) and an element of \( R_{v+1} \), that is, \( E_1 \cup E_2 \) holds, then \( f \) is \( \epsilon \)-close to having image diameter at most \( r \). This is because by redefining \( f \) only on points in \( (L_{u-1} \cup R_{v+1}) \) whose \( f \)-values are smaller or larger than \( f \)-values on all the samples, we get a function with image diameter at most \( r \). Moreover, an application of the union bound and the fact that each of \( \Pr(L_{u-1}) \) and \( \Pr(R_{v+1}) \) is at most \( \epsilon/2 \), gives \( \Pr(L_{u-1} \cup R_{v+1}) \leq \epsilon \). It remains to show that \( \Pr(E_1 \cup E_2) \) is at most \( 1/4 \). By definition of \( E_1 \), we have \( \Pr(E_1) = (1 -
\[ \Pr(L_{u-1})^s \leq (1 - \epsilon/2)^s, \] where we used the fact that \( \Pr(L_{u-1}) \leq \epsilon/2 \). Analogously, we get \( \Pr(E_2) \leq (1 - \epsilon/2)^s \). Applying the union bound, we get \( \Pr(E_1 \cup E_2) \leq 2 \cdot (1 - \epsilon/2)^s \). Since \( s \) is \( \lceil \frac{6}{\epsilon} \rceil \), the latter is at most \( 1/6 \), as required.

Proof of Lemma 3.2 In this proof, all references to distance to a property (e.g., \( \epsilon \)-far from the Lipschitz property, \( \epsilon \)-close to having image diameter \( r \) etc.) are with respect to distribution \( \Pi \) on \( D \). (See Definition 2.1.) The required tester is Algorithm 1.

Algorithm 1: Test \( (f : D \to R, \epsilon) \)

1. Let \( r \leftarrow \text{Sample-diameter}(f, \epsilon/2) \). If \( r > \text{diam}(D) \), reject.
2. Let \( A(f, \epsilon, r) \) be the Lipschitz tester as in Lemma 3.2. Namely, it gets oracle access to function \( f : D \to R \) and receives a parameter \( r \) (in addition to \( \epsilon \)) such that \( r \geq \text{ImD}(f) \).
3. Run \( A(f, \epsilon/2, r) \) twice and accept if both runs of the algorithm accept; otherwise, reject.

Algorithm 1 always accepts a Lipschitz function. Consider a function \( f \) which is \( \epsilon \)-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 3.1, \( \Pr[E] \leq 1/4 \). If \( r > \text{diam}(D) \) then the tester correctly rejects on line 1 because, by Claim 3.1, \( r \leq \text{ImD}(f) \), and a Lipschitz function on \( D \) must have image diameter at most \( \text{diam}(D) \).

It remains to consider the case when \( r \leq \text{diam}(D) \). Conditioned on \( E \) not happening (denoted \( \overline{E} \)), there is a function \( h \) with \( \text{ImD}(h) \leq r \) such that \( \text{dist}_\Pi(f, h) < \epsilon/2 \), where \( \text{dist}_\Pi(f, h) \) is as defined in Definition 2.1. 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 \( \text{ImD}(g) \leq r \) and \( \text{dist}_\Pi(f, g) < \epsilon/2 \). Since \( f \) is \( \epsilon \)-far from the Lipschitz property, by the triangle inequality, we get that \( g \) is \((\epsilon/2)\)-far from being Lipschitz. This fact together with the observation that \( g \) has image diameter at most \( r \) implies \( \Pr[A(g, \epsilon/2, r) \text{ rejects}] \)
is at least $\frac{2}{3}$.

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

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

$$\Pr[\text{Step 3 rejects } f] \geq \Pr[\text{Step 3 rejects } \mid \bar{E}] \cdot \Pr[\bar{E}] \geq \frac{8}{9} \cdot \frac{5}{6} \geq \frac{2}{3}.$$  

Finally, observe that the running time is $O(1/\epsilon)$ if $\text{SAMPLE-DIAMETER}$ returns $r > \text{diam}(D)$. Otherwise, it is $2 \cdot T(\epsilon/2, r)$, where $T(\epsilon/2, r)$ is the running time of $A(f, \epsilon/2, r)$ and $r \leq \min\{\text{diam}(D), \text{Im}D(f)\}$.

### 3.2 A Lower bound on query complexity of the Lipschitz tester for the hypercube

In this section, we give a lower bound on the query complexity of the tester for the hypercube which matches the upper bound in Theorem 3.1 for the case of the $\{0, 1, 2\}$ range, constant $\epsilon$ and $\delta = 1$.

**Theorem 3.2.** 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 the above theorem using the method presented by Blais, Brody, and Matulef [BBM12] of reducing a suitable communication complexity problem to the testing problem. In [BBM12], 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.

**Proof of Theorem 3.2** Consider the following communication game between Alice and Bob in the public randomness model, where the players generate messages based
Claim 3.2. Given subsets both to compute \( DISJ \) if \( 1 \) and they would like to compute the set-disjointness function \( DISJ \), the minimum number of bits Alice and Bob must communicate for them both to compute \( DISJ(A, B) \) with probability at least \( 2/3 \) on any input pair \((A, B)\), is \( \Omega(d) \).

Alice and Bob can reduce the problem of computing \( DISJ \) to testing the Lipschitz property as follows. Note that a set \( \Omega(d) \) 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 \( DISJ(A, B) \), which is \( 1 \) if \( A \cap B = \emptyset \) and \( 0 \) otherwise. It is well-known [KS92, BYJKS04, Raz92] that \( R(DISJ) \), the minimum number of bits Alice and Bob must communicate for them both to compute \( DISJ(A, B) \) with probability at least \( 2/3 \) on any input pair \((A, B)\), is \( \Omega(d) \).

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, \( R(DISJ) \leq 2q(d) \). The claimed bound then follows from the \( \Omega(d) \) bound on \( R(DISJ) \).

The following claim was used in the proof of Theorem 3.2.

Claim 3.2. 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. \( \square \)
Testing the Lipschitz property over unknown product distribution

In this chapter, we present a Lipschitz tester for functions defined on \( \{0, 1\}^d \) when the underlying distribution on \( \{0, 1\}^d \) is an unknown product distribution. Let \( \Pi = (p_1, \ldots, p_d) \) be a product distribution on \( \{0, 1\}^d \). Namely, a sample distributed according to \( \Pi \) has its \( i \)-th coordinate 1 with probability \( p_i \) and 0 with probability \( 1 - p_i \), independently, for each \( i \in [d] \). Our tester does not need to know \( p_i \)'s in order to test the Lipschitz property with respect to \( \Pi \). It only needs access to independent samples from \( \Pi \). In the analysis of the tester, however, we do make use of \( p_i \)'s. Given a vertex \( x \in \{0, 1\}^d \), we let \( p_x \) denote the probability mass of \( x \) under distribution \( \Pi \). Observe that \( p_x = \prod p_i^{x_i}(1 - p_i)^{(1 - x_i)} \) when \( \Pi = (p_1, \ldots, p_d) \). In this chapter, we give an algorithm for testing the Lipschitz property of functions on \( \mathcal{H}_d \) with respect to an arbitrary and unknown product distribution. Recall that \( \delta\mathbb{Z} \) denotes the set of integer multiples of \( \delta \) and \( ImD(f) \) denotes the image diameter of function \( f \), where the image diameter of a function is defined in Definition 2.7.

Theorem 4.1 (Lipschitz tester over product distribution for \( \delta\mathbb{Z} \)-valued functions). Fix \( \epsilon \in (0, 1) \) and \( \delta \in (0, \epsilon/d^2) \). There exists a non-adaptive tester for testing the Lipschitz property of functions \( f : \{0, 1\}^d \rightarrow \delta\mathbb{Z} \) with respect to an arbitrary (unknown) product distribution on \( \{0, 1\}^d \). The tester has 1-sided error and runs in time \( O(d \min\{d, \text{ImD}(f)\} \delta (\epsilon - d^2 \delta)) \).

The Lipschitz tester used to prove the above theorem is given in Section 4.1. Here, we give a high-level description. Our tester samples edges of the hypercube \( \mathcal{H}_d \) from
a fixed distribution on edges and rejects iff the selected edges contain a violated edge. (Recall the definition of a violated edge from Chapter 2.) The distribution on edges that we use assigns probability mass \((p_x + p_y)/d\) to each edge \(\{x, y\}\), where \(p_x\) (respectively, \(p_y\)) denotes the probability mass of \(x\) (respectively, \(y\)) under distribution \(\Pi\). Observe that this is indeed a distribution on edges, namely, the sum of \((p_x + p_y)/d\) over all edges \(\{x, y\}\) of the hypercube is 1. Since a Lipschitz function does not have any violated edges, the tester always accepts a Lipschitz function. Consider a function that is \(\epsilon\)-far from Lipschitz. (Recall that \(\epsilon\)-far is defined with respect to the distribution \(\Pi\).) Our main technical lemma gives a lower bound on the probability of sampling a violated edge from the distribution on edges that we just defined. We state the lemma here and prove it in Section 4.2.

**Lemma 4.1.** Let function \(f : \{0, 1\}^d \to \delta \mathbb{Z}\) be \(\epsilon\)-far from Lipschitz. Let \(V(f)\) denote the set of edges in \(H_d\) violated by \(f\). Then

\[
\sum_{(x,y) \in V(f)} \frac{(p_x + p_y)}{d} \geq \frac{\delta (\epsilon - d^2 \delta)}{d \cdot \text{ImD}(f)}.
\]

We also obtain the following corollary of Theorem 4.1. This extends the Lipschitz tester of Theorem 4.1 to real-valued functions.

**Corollary 4.1** (Lipschitz tester over product distribution for \(\mathbb{R}\)-valued functions). Fix \(\epsilon \in (0, 1)\) and \(\delta \in (0, \epsilon/d^2)\). There exists a \((1 + \delta)\)-approximate Lipschitz tester (see Definition 2.3) for testing the Lipschitz property of functions \(f : \{0, 1\}^d \to \mathbb{R}\) with respect to an arbitrary (unknown) product distribution on \(\{0, 1\}^d\). The tester has 1-sided error and runs in time \(O\left(\frac{d \min \{d, \text{ImD}(f)\}}{\delta (\epsilon - d^2 \delta)}\right)\).

Since the proof is identical to the proof of Corollary 3.1, we do not repeat it.

**4.1 Lipschitz tester over product distribution: proof of Theorem 4.1**

In this section, we prove Theorem 4.1. Lemma 3.2 implies that it is sufficient to give a Lipschitz tester which knows an upper bound on the image diameter of the input function \(f\). Algorithm 2 given below is the required tester.
Algorithm 2: Lipschitz tester for $H_d$

**Input**: Oracle access to function $f : \{0, 1\}^d \to \delta \mathbb{Z}$, oracle access to independent samples from the (unknown) product distribution $\Pi$ on $\{0, 1\}^d$, domain parameter $d$, proximity parameter $\epsilon$, discretization/approximation parameter $\delta$, and an upper bound $r$ on the image diameter of function $f$.

1. **Repeat** independently $(2dr)/(\delta(\epsilon - d^2 \delta))$ times:
   2. Sample an edge $\{x, y\}$ of $H_d$ as follows. Pick a sample $x$ from distribution $\Pi$ on $\{0, 1\}^d$ and let $y$ be a uniform neighbor of $x$ in $H_d$.
   3. If $f$ violates the selected edge $\{x, y\}$, i.e., $|f(x) - f(y)| > 1$, then **reject**.
   4. If none of the iterations above rejects, then **accept**.

**Proof of Theorem 4.1** If the input function $f$ is Lipschitz, then Algorithm 2 always accepts. This is because a Lipschitz function does not have any violated edges. Next consider the case when $f$ is $\epsilon$-far from Lipschitz. What is the probability of selecting a specific edge $\{x, y\}$ of the hypercube in Step 2? This is precisely $(p_x + p_y)/d$, where $p_x$ and $p_y$ are the probabilities of selecting $x$ and $y$ under distribution $\Pi$, respectively. Therefore, the probability that a single iteration of the repeat loop (Step 1) rejects $f$ is given by the sum of $(p_x + p_y)/d$ over all violated edges $\{x, y\}$ of the hypercube. Using Lemma 4.1 and noting that $r$ is an upper bound on the image diameter of $f$, we get that this sum is at least $k = (\delta(\epsilon - d^2 \delta))/(dr)$. Since the repeat loop runs $2/k$ times independently, by standard arguments, we get that $f$ is rejected with probability at least $2/3$, as required. The running time of the tester is $O(1/k)$. By Lemma 3.2, the running time stated in the theorem follows.

### 4.2 An asymmetric repair operator

In this section, fix a product distribution $\Pi = (p_1, \ldots, p_d)$ on $\{0, 1\}^d$. Our proof of Lemma 4.1 follows closely the proof of Lemma 3.1 of Chapter 3. Next we define an asymmetric basic operator. This can be seen as an asymmetric version of the basic operator (see Definition 6.2) of Chapter 3.

**Definition 4.1** (Asymmetric basic operator). Given $f : \{0, 1\}^d \to \delta \mathbb{Z}$, for each violated edge $\{x, y\}$ along dimension $i$, where $f(x) < f(y) - 1$, define $B_i$ as follows.
1. If $|x| > |y|$, then $B_i[f](x) = f(x) + (1 - p_i)\delta$ and $B_i[f](y) = f(y) - p_i\delta$.

2. If $|x| < |y|$, then $B_i[f](x) = f(x) + p_i\delta$ and $B_i[f](y) = f(y) - (1 - p_i)\delta$.

Now we define the repair operator which plays the same role as the averaging operator (see Definition 3.3) of Chapter 3.

**Definition 4.2** (Repair operator). Let $f'$ be the function obtained from $f$ by applying $B_i$ repeatedly until there are no violated edges along the $i$-th dimension. Then, $A_i[f]$ is the function obtained by rounding the function values of $f'$ to the closest $\delta\mathbb{Z}$ values.

Next we generalize the definition of violation score (see Definition 3.5) of Chapter 3. We use the same notation as used in Chapter 3.

**Definition 4.3.** The violation score of an edge with respect to function $f$, denoted by $vs(\{x, y\})$, is $\max(0, (p_x + p_y)(|f(x) - f(y)| - 1))$. The violation score along dimension $i$, denoted by $VS^i(f)$, is the sum of violation scores of all edges along dimension $i$.

Observe that an edge has a positive violation score if and only if it is violated. Moreover, since $f$ is $\delta\mathbb{Z}$-valued, the violation score of a violated edge $\{x, y\}$ is at least $\delta \cdot (p_x + p_y)$. Also, noting that $|f(x) - f(y)| - 1$ is at most $\text{ImD}(f)$, we get that the violation score of an edge $\{x, y\}$ is never greater than $(p_x + p_y) \cdot \text{ImD}(f)$. These observations imply the following inequality (which corresponds to Inequality 3.1 of Chapter 3).

$$\delta \cdot \sum_{\{x, y\} \in \mathcal{V}^i(f)} (p_x + p_y) \leq VS^i(f) \leq \sum_{\{x, y\} \in \mathcal{V}^i(f)} (p_x + p_y) \cdot \text{ImD}(f). \quad (4.1)$$

Lemma 4.2 below is the analogue of Lemma 3.3 from Chapter 3. It shows that the repair operator $A_i$ does not introduce violated edges along a fully repaired dimension. While it may increase a violation score along an unrepaired dimension, the increase is at most $\delta$.

**Lemma 4.2.** For all $i, j \in [d]$, where $i \neq j$, and every function $f : \{0, 1\}^d \rightarrow \delta\mathbb{Z}$, the following holds.

- **(progress)** Applying the repair operator $A_i$ does not introduce new violated edges in dimension $j$ provided dimension $j$ dimension $j$ did not have any violated edges to begin with. Namely, $VS_j(f) = 0 \Rightarrow VS_j(A_i[f]) = 0$.  

• **(accounting)** Applying the repair operator \( A_i \) does not increase the violation score in dimension \( j \) by more than \( \delta \), i.e. \( \text{VS}_j(A_i[f]) \leq \text{VS}_j(f) + \delta \).

Application of the repair operator \( A_i \) entails several applications of the basic operator \( B_i \) followed by a rounding step. In Lemma 4.3, we show that the repeated applications of \( B_i \) does not increase the violation score along other dimensions. While the rounding step that follows may potentially increase the violation score along other dimensions, in Claim 4.1, we show that the damage due to this step is limited: no new violated edges are created along a fully repaired dimension and the violation score along an unrepaired dimension can increase by at most \( \delta \). Finally, we use Lemma 4.3 and Claim 4.1 to prove Lemma 4.2.

**Lemma 4.3.** Suppose \( f : \mathcal{H}_d \to \mathbb{R} \) is such that for every edge \( \{x, y\} \) in \( \mathcal{H}_d \), the difference \( f(x) - f(y) \in \delta \mathbb{Z} \). Let \( f' \) be the function obtained from \( f \) by applying \( B_i \) repeatedly until there are no violated edges along the \( i \)-th dimension. Then for every dimension \( j \neq i \), \( \text{VS}_j(f') \leq \text{VS}_j(f) \).

**Proof.** First observe that it is sufficient to prove the lemma for a single application of the basic operator \( B_i \). This is because for every edge \( \{x, y\} \), the following holds: \( f(x) - f(y) \in \delta \mathbb{Z} \Rightarrow B_i[f](x) - B_i[f](y) \in \delta \mathbb{Z} \). To see this observe that, by definition of the basic operator, \( B_i[f](x) - B_i[f](y) \) is either \( f(x) - f(y) + \delta \) or \( f(x) - f(y) - \delta \) and we already started with the assumption that \( f(x) - f(y) \in \delta \mathbb{Z} \). Next we prove the lemma for one step of the basic operator.

We follow the proof outline of Lemma 3.1 of Chapter 3. As shown there, it is enough to analyze the effect of applying \( B_i \) on one fixed disjoint square formed by adjacent edges that cross dimensions \( i \) and \( j \). (This is because edges along dimensions \( i \) and \( j \) form disjoint squares in the hypercube. So having proved the statement for one fixed square of the hypercube, the full claim follows by summing up the inequalities over all such squares.)

Consider the two dimensional function \( f : \{x_b, x_t, y_b, y_t\} \to \mathbb{R} \) where \( \{x_b, x_t, y_b, y_t\} \) are as positioned in the figure. Assume that the basic operator is applied along the
Combining the above inequalities, we get modified by the basic operator. Since \(v_s(\{x_b, x_t\})\) increases, \(f(x_t) > f(x_b) + 1 - p_t \delta\). Combining the above inequalities, we get \(f(y_t) \geq f(x_t) + 1 + \delta > f(x_b) + 2 + (1 - p_t) \delta \geq f(y_b) + 1 + (1 - p_t) \delta > f(y_b) + 1\). Thus the violation score increases along \(\{x_t, x_b\}\) by \((p_{x_b} + p_{x_t}) p_t \delta\) and decreases along \(\{y_b, y_t\}\) by \((p_{y_b} + p_{y_t}) (1 - p_t) \delta = (p_{x_b} + p_{x_t}) (p_{x_b} + p_{x_t}) (1 - p_t) \delta\) which is same as \((p_{x_b} + p_{x_t}) p_t \delta\), keeping the violation score along the dimension \(j\) unchanged.

If the bottom edge is violated, then the increase in \(v_s(\{x_b, x_t\})\) implies that \(f(x_b)\) must decrease (after application of \(B_i\)) by \(p_t \delta\) (since \(|x_b| < |y_b|\) implying \(f(y_b) + 1 < f(x_b)\)). Therefore \(f(x_t) > f(x_t) + 1 > f(x_b) + 2 - 2 p_t \delta \geq f(y_b) + 3 - 2 p_t \delta + \delta \geq f(y_b) + 1 + \delta\). The last inequality is true since \(\delta \leq 1\) and \(p_t \leq 1\). Thus, \(v_s(\{x_t, x_b\})\) increases by at most \((p_{x_b} + p_{x_t}) 2 p_t \delta\) while \(v_s(\{y_t, y_b\})\) decreases by \((p_{y_b} + p_{y_t}) (1 - p_t) \delta = (p_{x_b} + p_{x_t}) 2 p_t \delta\), ensuring that violation score along the vertical dimension does not increase.

Now we turn to the case when \(B_i[f(x_t)] < B_i[f(x_b)]\). By the arguments very similar to the first case, it can be proved that \(f(x_t) \geq f(y_t) + 1 + \delta\) and the application of basic operator decreases \(f(x_t)\) by \(p_t \delta\) and increases \(f(y_t)\) by \((1 - p_t) \delta\).

If the bottom edge is not violated then \(f(y_b) \geq f(x_b) - 1\) and \(f(x_b)\) and \(f(y_b)\) are not modified by the basic operator. Since \(v_s(\{x_t, x_b\})\) increases, \(f(x_b) > f(x_t) + 1 - p_t \delta\). Combining the above inequalities, we get \(f(y_b) \geq f(x_b) - 1 > f(x_t) - p_t \delta \geq f(y_b) + 1 + \delta(1 - p_t)\). Thus the violation score increases along \(\{x_t, x_b\}\) by \((p_{x_b} + p_{x_t}) p_t \delta\) and
decreases along \(\{y_b, y_t\}\) by \((p_{y_b} + p_{y_t})(1 - p_t)\delta = (p_{x_b} + p_{x_t}) \left(\frac{p_t}{1 - p_t}\right)(1 - p_t)\delta\) which is same as \((p_{x_b} + p_{x_t})p_t\delta\), keeping the violation score along the dimension \(j\) unchanged.

If the bottom edge is violated, then the increase in \(vs\{x_b, x_t\}\) implies that \(f(x_b)\) must increase implying \(f(y_b) > f(x_b) + 1\). Therefore, the increase in \(vs\{x_b, x_t\}\) implies that \(f(x_b) + p_t\delta > f(x_t) - p_t\delta + 1\) or \(f(x_b) > f(x_t) - 2p_t\delta + 1\). Combining the above inequalities, we get \(f(y_b) > f(x_b) + 1 > f(x_t) - 2p_t\delta + 2 \geq f(y_t) + 3 + \delta - 2p_t\delta \geq f(y_t) + 1 + \delta\). The last inequality is true since \(\delta \leq 1\) and \(p_t \leq 1\). Thus, \(vs\{x_t, x_b\}\) increases by at most \((p_{x_b} + p_{x_t})2p_t\delta\) while \(vs\{y_t, y_b\}\) decreases by \((p_{y_t} + p_{y_b})2(1 - p_t)\delta = (p_{x_b} + p_{x_t})2p_t\delta\), ensuring that the violation score along the vertical dimension does not increase.

**Claim 4.1** (Rounding is safe). Given \(a, b \in \mathbb{R}\) satisfying \(|a - b| \leq 1\), let \(a'\) (respectively, \(b'\)) be the value obtained by rounding \(a\) (respectively, \(b\)) to the closest \(\delta \mathbb{Z}\) integer. Then \(|a' - b'| \leq 1\).

**Proof.** Without loss of generality assume \(a \leq b\). For \(x \in \mathbb{R}\), let \([x]_{\delta}\) be the largest value in \(\delta \mathbb{Z}\) not greater than \(x\). Observe that \(a' \in \{[a]_{\delta}, [a]_{\delta} + \delta\}\). Using the fact that \([a]_{\delta} \leq b' \leq [a]_{\delta} + 1 + \delta\), we see that if \(a' = [a]_{\delta} + \delta\) then \(|b' - a'| \leq 1\) always holds. Therefore, assume \(a' = [a]_{\delta}\). This can happen only if \(a \leq [a]_{\delta} + \delta/2\). The latter implies \(b \leq [a]_{\delta} + 1 + \delta/2\) (using the fact that \(b - a \leq 1\)). That is \(b' = [a]_{\delta} + 1 + \delta\). In other words, \(b' \leq [a]_{\delta} + 1\) again implying \(|b' - a'| \leq 1\), as required.

**Proof of Lemma 4.2** Let \(f'\) be the function from the statement of Lemma 4.3. Then function \(A_i[f]\) is obtained by rounding the values of \(f'\) to the closest \(\delta \mathbb{Z}\) values. Since rounding can never create new edge violations by Claim 4.1, we immediately get the first part of the lemma. The second part follows from the observation that the rounding step modifies each function value by at most \(\delta/2\). Correspondingly, the violation score of an edge along the \(j\)-th dimension changes by at most \(2 \cdot (\delta/2) \cdot (p_u + p_v)\) where the factor 2 comes because both endpoints of an edge may be rounded. Summing over all edges in the \(j\)-th dimension, we get, increase in violation score \(\leq \sum_{\{u,v\}} \delta \cdot (p_u + p_v) = \delta\) where the last equality holds because edges along the \(j\)-th dimension form a perfect matching and therefore the probabilities \(p_u + p_v\) sum to 1.
Proof of Lemma 4.1  Let \( f_0 = f \) and for each \( i \in [d] \), let \( f_i = A_i[f_{i-1}] \) where \( A_i \) is the repair operator defined in Definition 4.2. In effect, we have the following picture.

\[
f = f_0 \xrightarrow{A_1} f_1 \xrightarrow{A_2} f_2 \rightarrow \cdots \rightarrow f_{d-1} \xrightarrow{A_d} f_d.
\]

Fix \( i \in [d] \). By definition of \( A_i \), we know that, only the endpoints of violated edges of \( f_{i-1} \) along dimension \( i \) are modified to obtain \( f_i \). This gives the first inequality in the following calculation. The second and fourth inequalities follow from (4.1) and the third inequality follows from Lemma 4.2.

\[
dist_{\Pi}(f_{i-1}, f_i) = dist_{\Pi}(f_{i-1}, f_i) \leq \sum_{(x,y) \in V_i(f_{i-1})} (p_x + p_y) \leq \frac{1}{\delta} \cdot VS^i(f_{i-1})
\]

\[
\leq (i - 1) \cdot \delta + \frac{1}{\delta} \cdot VS^i(f)
\]

\[
\leq (i - 1) \cdot \delta + \frac{ImD(f)}{\delta} \sum_{(x,y) \in V^i(f)} (p_x + p_y).
\]

By triangle inequality, we know, \( dist_{\Pi}(f, f_d) \) is at most the sum of \( dist_{\Pi}(f_{i-1}, f_i) \) over all \( i \in [d] \). Using the bound on the latter quantity obtained above, we get the following.

\[
dist_{\Pi}(f, f_d)
\]

\[
\leq \sum_{i \in [d]} ((i - 1) \cdot \delta + \frac{ImD(f)}{\delta} \sum_{i \in [d]} \sum_{(x,y) \in V^i(f)} (p_x + p_y)
\]

\[
= \frac{(d^2 - d) \cdot \delta}{2} + \frac{ImD(f)}{\delta} \sum_{(x,y) \in V(f)} (p_x + p_y)
\]

(4.2)

Since \( f \) is \( \epsilon \)-far from being Lipschitz, we have \( dist_{\Pi}(f, f_d) \geq \epsilon \). Therefore, using (4.2) and the fact that \( (d^2 - d)/2 \) is at most \( d^2 \), we have

\[
\sum_{(x,y) \in V(f)} \frac{(p_x + p_y)}{d} \geq \frac{\delta(\epsilon - d^2 \delta)}{d \cdot ImD(f)},
\]

as required. \( \square \)
Chapter 5

Testing edge-transitive properties that allow extension

5.1 Overview

In this chapter, 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 $\overrightarrow{L}_n$. Observe that the Lipschitz properties on $L_n$ and on $\overrightarrow{L}_n$ 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 $L_n$ is $y - x$. For $\overrightarrow{L}_n$, the shortest path distance from $x$ to $y$ is $y - x$, and from $y$ to $x$ is $\infty$. 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 [BGJ+12b], based on 2-transitive-closure spanners (see Definition 2.9), 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. 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\(^1\), whenever $(x, y)$ and $(y, z)$ satisfy the above conditions, so does $(x, z)$; and

\(^1\) $\prec$ 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 $G_n$. 
(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 5.2.) Examples of edge-transitive properties include the $c$-Lipschitz property on directed hypergrids and variants of monotonicity. (See the discussion after Definition 5.2.)

The Lipschitz property for functions on $f : L_n \rightarrow R$ allows extension for most ranges $R$ of interest. We characterize such ranges $R$ (in Claim 5.2) as discretely metrically convex metric spaces. Metric convexity is a standard notion in geometric functional analysis (see, e.g. [BL00]). 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 5.1** (Definition 1.3 of [BL00] 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) $\alpha$ and $\beta$ 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 5.2, gives an efficient tester for every edge-transitive property that allows extension and, in particular, applies to the Lipschitz property of functions $f : L_n \rightarrow R$, where $R$ is discretely metrically convex.

**Theorem 5.1.** 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 $f : G_n \rightarrow R$ 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 $L_n$ and $L^\rightarrow_n$ are identical. We therefore get the following corollary.

**Corollary 5.1.** The Lipschitz property of functions $f : L_n \rightarrow R$ for every discretely metrically convex space $R$ can be tested in time $O(\frac{\log n}{\epsilon})$. 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 \( \text{ImD}(f) \) is small, the following theorem, proved in Section 5.2, gives a faster tester than Corollary 5.1.

**Theorem 5.2.** The Lipschitz property of functions \( f : \mathcal{L}_n \rightarrow \mathbb{R} \) can be tested nonadaptively and with one-sided error in time \( O\left( \log \min\{n, \text{ImD}(f)\} \right) \).

Our next theorem, proved in Section 5.4, shows that the upper bound of Theorem 5.2 is tight for nonadaptive one-sided error testers. Even though it is stated for range \([r]\) for concreteness, as shown in Corollary 5.2 at the end of Section 5.4, 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 5.1.

**Theorem 5.3.** Every nonadaptive one-sided error tester of the Lipschitz property of functions \( f : \mathcal{L}_n \rightarrow [r] \) must make \( \Omega(\log \min\{n, r\}) \) queries.

To prove Theorem 5.3, 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 \( r \in [n] \) 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 \( \Delta f \), defined by \( \Delta f(1) = 0 \) and \( \Delta f(x) = f(x) - f(x - 1) \) for all \( x \geq 2 \).

### 5.2 A 2-TC-spanner based tester: proof of Theorem 5.1

In this section, we prove Theorem 5.1. We start by giving the definition and examples of edge-transitive properties that allow extension. Recall the definition of comparable and incomparable elements (Definition 2.8).

**Definition 5.2.** Let \( G \) be a directed graph and \( R \) be an arbitrary range. A property \( \mathcal{P} \) of functions \( f : G \rightarrow R \) 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 $f(x) \leq c \cdot d_G(x, y) \cdot f(y)$ 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 $\overrightarrow{H}_d$ and the hypergrid $\overrightarrow{H}_{n,d}$. 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 5.2. Fortunately, when the domain of functions is \( \overrightarrow{L}_n \), for many ranges of interest, such as \( \mathbb{R}^k \), equipped with \( \ell_1, \ell_2 \) or \( \ell_\infty \), \( \mathbb{Z}^k \), equipped with \( \ell_1 \) or \( \ell_\infty \), 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 5.2.

Next we prove Theorem 5.1 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 [BGJ+12b] for functions \( f : G_n \to \mathbb{R} \), based on 2-TC-spanners (see Definition 2.9).

**Proof of Theorem 5.1** 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 \( \epsilon \)-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 \( \epsilon \)-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. \( \square \)
5.3 Line: testing if a function on $\mathcal{L}_n$ is Lipschitz

In this section, we first prove Corollary 5.1 that gives a tester for the Lipschitz property of functions $f : \mathcal{L}_n \to R$ for every discretely metrically convex space $R$. Later we prove Theorem 5.2 which strengthens Corollary 5.1 for real-valued functions.

**Proof of Corollary 5.1** Since the Lipschitz properties on $\mathcal{L}_n$ and $\mathcal{L}_n^\rightarrow$ are identical, we prove the statement for the latter. A sparse 2-TC-spanner of the directed line $\mathcal{L}_n^\rightarrow$ 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 [Yao82, CFL85, CFL83, AS87, Cha87, BTS94, Tho97, DGL+99]. It is surveyed as a stand-alone construction in [Ras10]. We review this construction here, since it is used later, in the proof of Theorem 5.2. 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 5.2, allows extension whenever $R$ is discretely metrically convex, Theorem 5.1 implies the first part of the corollary. Then the second part, with the examples of spaces $R$, follows from Claim 5.3.

The strengthening of Corollary 5.1 to Theorem 5.2 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 3.1.

**Proof of Theorem 5.2** We begin by proving the following claim.

**Claim 5.1.** Let function $f : [n] \to \mathbb{R}$ be $\epsilon$-far from Lipschitz. Let $H = ([n], E)$ be the 2-TC-spanner of the line graph constructed in the proof of Corollary 5.1. Let $E'$ be the subset of $E$ consisting of edges $(x, y)$ satisfying $|x - y| < \text{ImD}(f)$. Then the fraction of edges in $E'$ violated by $f$ is at least $\frac{\epsilon}{10 \log \min\{\text{ImD}(f), n\}}$. 
Proof. From the proof of Theorem 5.1, 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 $|x - y| \geq \text{ImD}(f)$ must satisfy $|f(x) - f(y)| \leq \text{ImD}(f) \leq |x - y|$. Thus, $E'$ has at least $\epsilon n/2$ violated edges.

It remains to show that $|E'| \leq 5n \log r$, where $r = \min \{\text{ImD}(f), n\}$. 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 5.1. In the $i$th 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 $s = \lceil 6 \log \min \{\text{ImD}(f), n\} / \epsilon \rceil$ edges uniformly and independently at random from the set $E'$ defined in Claim 5.1. 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 5.1, functions which are $\epsilon$-far from Lipschitz are rejected with probability at least $2/3$. This tester, together with Lemma 3.2, implies the tester in the statement of Theorem 5.2, that works without the knowledge of $\text{ImD}(f)$. 

We finish this section by proving two claims used in the proof of Corollary 5.1. Claim 5.2 characterizes ranges $R$ for which the Lipschitz property of functions $f : [n] \to R$ allows extension. Claim 5.3 gives examples of such ranges.

Claim 5.2. 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 $\ell$ (respectively, $r$) be the $D'$-vertex closest to $x$ on the left (respectively, right). If either $\ell$ 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 2.6) because for every $y \in D'$, one of $\ell$ or $r$ lies on the shortest path between $x$ and $y$. If one of $\ell$ 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 $d_R(f(x), f(r)) \leq r - x$, 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 $\alpha$ and $\beta$. Now, consider a partial function $f : [\alpha + \beta + 1] \rightarrow R$ 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. \hfill \Box

**Claim 5.3** (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 $u \in \mathbb{R}^k$ and $p \in [1, \infty)$, let $||u||_p = (\sum_{i \in [k]} |u_i|^p)^{1/p}$ denote the $\ell_p$-norm of $u$. The $\ell_p$ metric on $\mathbb{R}^k$ is defined by $d_{\ell_p}(u, v) = ||u - v||_p$. Given elements $u, v \in \mathbb{R}^k$ and positive real numbers $\alpha$ and $\beta$ satisfying $||u - v||_p \leq \alpha + \beta$, let $w = \frac{\alpha v + \beta u}{\alpha + \beta}$. Then $d_{\ell_p}(u, w) \leq \alpha$ and $d_{\ell_p}(w, v) \leq \beta$. This shows that $(\mathbb{R}^k, \ell_p)$ for $p \in [1, \infty)$ is metrically convex. Fact 2.1 and Claim 5.2 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 $\alpha$ and $\beta$. If $\alpha \geq d_G(u, v)$, then trivially $w = v$ satisfies $d_G(u, w) \leq \alpha$ and $d_G(w, v) \leq \beta$. Otherwise, let $w$ be the vertex at distance $\alpha$ from $u$ on a shortest path between $u$ and $v$. Such a vertex exists because $\alpha < d_G(u, v)$. Then $d_G(u, w) + d_G(w, v) = d_G(u, v) \leq \alpha + \beta$ implies $d_G(w, v) \leq \beta$. Thus, $w$ is the required vertex. In particular, $\mathbb{Z}^k$ equipped with $\ell_1$ 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 \(\ell_1\) metric and \(\ell_\infty\) metric are identical on \(\mathbb{Z}\). Specifically, suppose \(u, v \in \mathbb{Z}^k\) and \(\|u - v\|_\infty \leq \alpha + \beta\) for positive integers \(\alpha\) and \(\beta\). Then, by the definition of \(\ell_\infty\), we have \(\max_{j \in [k]} |u_j - v_j| \leq \alpha + \beta\). Therefore, \(|u_j - v_j| \leq \alpha + \beta\) 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 \(|u_j - w_j| \leq \alpha\) and \(|w_j - v_j| \leq \beta\). Define \(w\) by \(w_j = w_j\). Then \(\|u - w\|_\infty = \max_{j \in [k]} |u_j - w_j| \leq \max_{j \in [k]} \alpha \leq \alpha\). Similarly, \(\|w - v\|_\infty \leq \beta\), thus proving that \((\mathbb{Z}^k, \ell_\infty)\) is discretely metrically convex.

\[\square\]

### 5.4 A Lower bound on testing the Lipschitz property on the line

In this section, we prove Theorem 5.3.

**Proof of Theorem 5.3** 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 2.1, every such function can be extended to a Lipschitz function. To prove our lower bound, we use Yao’s principle. Namely, we define a distribution \(\mathcal{N}\) on input functions \(f: [n] \to [r]\) 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 \(1/3\).

![Figure 5.1. The discrete derivative function, \(\Delta f_i\).](image)

Fix positive integers \(n, r \geq 8\). 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 \(2^{t+1} \leq r\).
Fix $i \in [\ell - 2]$. We describe the function $f_i$ by giving its \textit{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 $x \geq 2$. 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 $\Delta f_i$ is $\Sigma = \{-2, -1, 0, 1, 2\}$. To define $\Delta f_i$, consider the string $\sigma$ over the alphabet $\Sigma$ defined by

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

where exponentiation denotes repetition, namely, $a^b$ means $a$ repeated $b$ times. Then $\sigma$ 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 5.1.)

Now we show that the resulting functions $f_i$ are 1/4-far from Lipschitz. Let $P_i$ denote the partition of $[n]$ into intervals of size $2^i$. Namely, $P_i$ consists of intervals $I_j^i = [1 + (j - 1)2^i, j2^i]$ for all $j \in [n/2^i]$. Let $L_j^i = (1 + (j - 1)2^i, j2^i]$ and $R_j^i = [1 + (j - 1)2^i, j2^i]$ denote the half-open intervals corresponding to $I_j^i$. Observe that for each odd $j$, all pairs $(x, y) \in L_j^i \times R_{j+1}^i$ are violated by $f_i$. Therefore, to make $f_i$ Lipschitz, we have to redefine it on all points in $L_j^i$ or on all points in $R_{j+1}^i$, that is, on at least 1/4 of points in $I_j^i \cup I_{j+1}^i$. 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 $x \in L_j^i$ and $y \in R_{j+1}^i$ 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 $\mathcal{N}$, $\Pr[f$ violates $(a_s, a_{s+1})] \leq 1/|\mathcal{F}_\ell|$ for each $s \in [q - 1]$. By the union bound,

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

When $q \leq 2t/3$, this is less than $2/3$. That is, every deterministic one-sided error non-adaptive 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 $\ell = \lceil \log n \rceil$. 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 $x \in [2^i]$. Moreover, for $x \geq \lceil \log n \rceil + 1$, $\tilde{f}_i(x)$ is defined to be $f_i(2^i)$. 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 5.3 to all discretely metrically convex spaces.

**Corollary 5.2.** 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 $f : L_n \to R$ must make $\Omega(\log \min\{n, r\})$ queries.

The corollary follows from the following claim that states that the line $L_r$ (used as the range space in Theorem 5.3) can be embedded into $R$ without distortion if $R$ is as stated in Corollary 5.2.

**Claim 5.4.** 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, $L_r$ 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. \qed
Testing if a function on $\mathcal{H}_{n,d}$ is Lipschitz

6.1 Overview

In this chapter, we present efficient testers for the Lipschitz property of functions $f : [n]^d \to \delta \mathbb{Z}$, where $\delta \in (0, 1]$ and $\delta \mathbb{Z}$ is the set of integer multiples of $\delta$. Points in the domain $[n]^d$ can be thought of as vertices of a $d$-dimensional hypergrid, where every pair of points at $\ell_1$ distance 1 is connected by an edge. Each edge $(x, y)$ imposes a constraint $|f(x) - f(y)| \leq c$ and a function $f$ is $c$-Lipschitz iff every edge constraint is satisfied.

6.1.1 Our results

We present two efficient testers of the Lipschitz property of functions of the form $f : [n]^d \to \delta \mathbb{Z}$ with running time polynomial in $d, n$ and $(\delta \epsilon)^{-1}$. Our testers are faster for functions whose image has small diameter. Recall from Definition 2.7 that the image diameter of $f$, denoted $\text{ImD}(f)$, is the difference between the maximum and the minimum values attained by $f$. The first tester has 1-sided error, that is, it always accepts Lipschitz functions. The second tester is faster (when $\sqrt{d} \gg \log(1/\epsilon)$ and $\text{ImD}(f)$ is large), but has 2-sided error, that is, it can err with probability at most 1/3 on both positive and negative instances.
Theorem 6.1 (Hypergrid tester). For all $\delta, \epsilon \in (0, 1]$, the Lipschitz property of functions $f : [n]^d \rightarrow \delta \mathbb{Z}$ can be tested nonadaptively with the following time complexity:

1. in $O \left( \frac{d}{\delta \epsilon} \cdot \log \min \{ \text{ImD} (f), n d \} \cdot \log \min \{ \text{ImD} (f), n \} \right)$ time with 1-sided error.
2. in $O \left( \frac{d}{\delta \epsilon} \cdot \log \min \{ \text{ImD} (f), n \sqrt{d \log(1/\epsilon)} \} \cdot \log \min \{ \text{ImD} (f), n \} \right)$ time with 2-sided error.

If the image diameter, $\delta$ and $\epsilon$ are constant, then both testers run in $O(d)$ time. This is tight already for the range $\{0, 1, 2\}$, even for the special case of the hypercube domain. This follows from Theorem 3.2.

6.1.2 Our techniques

For clarity of presentation, we first state and prove all our theorems for $\delta = 1$, i.e., for integer-valued functions. In Section 6.5, by discretizing (as was done in Chapter 3), we extend our results to the range $\delta \mathbb{Z}$.

The main challenge in designing a tester for functions on the hypergrid domains is avoiding an exponential dependence on the dimension $d$. We achieve this via a dimension reduction, i.e., a reduction from testing functions on the hypergrid $[n]^d$ to testing functions on the line $[n]$, that incurs only an $O(d \cdot \min \{ \text{ImD}, nd \})$ multiplicative overhead in the running time. In order to do this, we relate the distance to the Lipschitz property of a function $f$ on the hypergrid to the average distance to the Lipschitz property of restrictions of $f$ to 1-dimensional (axis-parallel) lines. For $i \in [d]$, let $e^i \in [n]^d$ be 1 on the $i$th coordinate and 0 on the remaining coordinates. Then for every dimension $i \in [d]$ and $\alpha \in [n]^d$ with $\alpha_i = 0$, the line $g$ of $f$ along dimension $i$ with position $\alpha$ is the restriction of $f$ defined by $g(x_i) = f(\alpha + x_i \cdot e^i)$, where $x_i$ ranges over $[n]$. We denote the set of lines of $f$ along dimension $i$ by $L_f^i$ and the set of all lines, i.e., $\bigcup_{i \in [d]} L_f^i$, by $L_f$. We denote the relative distance of a function $h$ to the Lipschitz property, i.e., the fraction of input points where the function needs to be changed in order to become Lipschitz, by $\epsilon^{Lip}(h)$. The technical core of our dimension reduction is the following theorem that demonstrates that if a function on the hypergrid is far from the Lipschitz property then a random line from $L_f$ is, in expectation, also far from it.

\footnote{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.}
**Theorem 6.2 (Dimension reduction).** For all functions \( f : [n]^d \to \mathbb{Z} \), the following holds:

\[
\mathbb{E}_{g \leftarrow L_f} [\epsilon_{\text{Lip}}(g)] \geq \frac{\epsilon_{\text{Lip}}(f)}{2 \cdot d \cdot \text{ImD}(f)}.
\]

To obtain this result, we introduce a smoothing procedure that “repairs” a function (i.e., makes it Lipschitz) one dimension at a time, while modifying it at a few points. Such procedures have been previously designed for restoring monotonicity of Boolean functions [GGL+00, DGL+99] and, as we saw (in Chapters 3 and 4), for restoring the Lipschitz property of functions on the hypercube domain. The key challenge is to find a smoothing procedure that satisfies the following three requirements: (1) It makes all lines along dimension \( i \) (i.e., in \( L_i^f \)) Lipschitz. (2) It changes only a small number of function values. (3) It does not make lines in other dimensions less Lipschitz, according to some measure. There are known smoothing operators (e.g., graph Laplacian) that make a function more Lipschitz [Oll09], but to the best of our knowledge there are no appropriate bounds on the number of function values that are changed.

**Smoothing Procedure for 1-dimensional Functions.** Our first technical contribution is a local smoothing procedure for functions \( f : [n] \to \mathbb{Z} \), which we call BubbleSmooth, in analogy to Bubble Sort. In one basic step, BubbleSmooth modifies two consecutive values (i.e., \( f(i) \) and \( f(i + 1) \) for some \( i \in [n - 1] \)) that violate the Lipschitz property, namely, differ by more than 1. It decreases the larger and increases the smaller by 1, i.e., it transfers a unit from the larger to the smaller. See Algorithm 3 for the description of the order in which basic steps are applied. BubbleSmooth is a natural generalization of the averaging operator of Chapter 3, used to repair an edge of the hypercube, that is essentially several applications of the basic step to the edge.

One challenge in analyzing BubbleSmooth is that when it is applied to all lines in one dimension, it may increase the average distance to the Lipschitz property for the lines in the remaining dimensions. Our second key technical insight is to use the \( \ell_1 \) distance to the Lipschitz property to measure the performance of our procedure on the line and its effect on other dimensions. The \( \ell_1 \) distance between functions \( f \) and \( f' \) on the same domain, denoted by \( |f - f'|_1 \), is the sum of \( |f(x) - f'(x)| \) over all values \( x \) in the domain. The \( \ell_1 \) distance of a function \( f \) to the nearest Lipschitz function over the same domain is denoted by \( \ell_1^{\text{Lip}}(f) \). Observe that the Hamming distance and the \( \ell_1 \) distance from a function to a property can differ by at most \( \text{ImD}(f) \). Later, we leverage
the fact that Lipschitz functions have a relatively small image diameter to relate the $\ell_1$ distance to the Hamming distance.

We prove that BubbleSmooth returns a Lipschitz function and that it makes at most twice as many changes in terms of $\ell_1$ distance as necessary to make a function Lipschitz.

**Theorem 6.3.** Consider a function $f : [n] \to \mathbb{Z}$ and let $f'$ be the function returned by BubbleSmooth$(f)$. Then (1) function $f'$ is Lipschitz and (2) $|f - f'|_1 \leq 2 \cdot \ell_1^{\text{Lip}}(f)$.

The proof of the second part of this theorem requires several technical insights. One of the challenges is that BubbleSmooth changes many function values, but then undoes most changes during subsequent steps. We define a transfer graph to keep track of the transfers that move a unit of function value during each basic step. Its vertex set is $[n]$ and an edge $(x, y)$ represents that a unit was transferred from $f(x)$ to $f(y)$. Since two transfers $(x, y)$ and $(y, z)$ are equivalent to a transfer $(x, z)$, we can merge the corresponding edges in the transfer graph, proceeding with such merges until no vertex in it has both incoming and outgoing edges. As a result, we get a transfer graph, where the number of edges, $|E|$, is twice the $\ell_1$ distance from the original to the final function.

To prove that $|E| \leq \ell_1^{\text{Lip}}(f)$, we show that the transfer graph has a matching with the violation score at least $|E|$. The violation score of an edge (or a pair) $(x, y)$ is the quantity by which $|f(x) - f(y)|$ exceeds the distance between $x$ and $y$. (Recall that $|f(x) - f(y)| \leq |x - y|$ for all Lipschitz functions $f$ on domain $[n]$.) The violation score of a matching is the sum of the violation scores over all edges in the matching. We observe (in Lemma 6.2) that $\ell_1^{\text{Lip}}(f)$ is bounded below by a violation score of any matching. The crucial step in obtaining a matching with a large violation score is pinpointing a provable, but strong enough property of the transfer graph that guarantees such a matching. Specifically, we show that the violation score of each edge in the graph is at least the number of edges adjacent to its endpoints at its (suitably defined) moment of creation (Lemma 6.1). For example, this statement is not true if we consider adjacent edges in the final transfer graph. The construction of a matching with a large violation score in the transfer graph is one of the key technical contributions of this paper. It is the focus of Section 6.2.

**Dimension Reduction with respect to $\ell_1$.** Our smoothing procedure for functions on the hypergrids applies BubbleSmooth to repair all lines in dimensions $1, 2, \ldots, d,$
one dimension at a time. We show that for all $i, j \in [d]$ applying \textbf{BubbleSmooth} in dimension $i$ does not increase the expected $\ell_1^{\text{Lip}}(f)$ for a random line $g$ in dimension $j$. The key feature of our smoothing procedure that makes the analysis tractable is that it can be broken down into steps, each consisting of one application of the basic step of \textbf{BubbleSmooth} to the same positions $(k, k+1)$ on all lines in a specific dimension. This allows us to show that one such step does not make other dimensions worse in terms of the $\ell_1$ distance to the Lipschitz property. The cleanest statement of the resulting dimension reduction is with respect to the $\ell_1$ distance.

**Theorem 6.4.** For all functions $f : [n]^d \rightarrow \mathbb{Z}$, we have: $\sum_{g \in L} \ell_1^{\text{Lip}}(g) \geq \frac{\ell_1^{\text{Lip}}(f)}{2}.$

**Our Testers and Effective Image Diameter.** The main component of our tester repeats the following procedure: Pick a line uniformly at random and run one step of the line tester. (We use the line tester from [JR11].) Our dimension reduction (Theorem 6.2) is crucial in analyzing this component. However, the bound in Theorem 6.2 depends on the image diameter of the function $f$. In the case of a non-Lipschitz function, it can be arbitrarily large, but for a Lipschitz function on $[n]^d$ it is at most the diameter of the space, namely $nd$ (notice this factor in part (1) of Theorem 6.1). In fact, for our application we can also use the observable diameter of the space [Gro99]: since the hypergrid exhibits Gaussian-type concentration of measure, one obtains that a Lipschitz function maps the vast majority of points to an interval of size $O(n\sqrt{d})$ (notice this factor in part (2) of Theorem 6.1). Our testers use a preliminary step to rule out functions with large image diameter (resulting in 1-sided error) or with large observable diameter (resulting in 2-sided error).

**Techniques.** Relating the distance to the property of a given function with the distance to the property of random restrictions has been successfully used to obtain testers for many properties. Notably, for functions on multi-dimensional domains, it has been done for testing low degree, monotonicity and, for the special case of the hypercube, the Lipschitz property. Two ideas that appeared repeatedly in proofs of this type of statements are self-correction (e.g., in low-degree testing) and repair (e.g., in monotonicity and Lipschitz testing). Specifically, in [GGL+00, DGL+99] and in Chapters 3 and 4, the function is repaired one dimension at a time. The procedure in [GGL+00, DGL+99], for repairing monotonicity of Boolean functions on the hypergrid domains sorts the 0-1
values on each line in a given dimension. There are at least three obstacles that make the design and analysis of our repair procedure significantly harder: (1) Our function values are not limited to 0s and 1s. (2) There is no natural unique Lipschitz function to which we should reconstruct (in the case of monotonicity, sorting gives such a function). (3) Unlike in the case of sorting, the Hamming distance does not work as a measure of progress for our operator.

The repair procedure in [DGL+99] for restoring monotonicity of functions on general ranges applies induction on the size of the range, using Boolean range as the base case. Observe that in the case of the Lipschitz property, functions with Boolean ranges are always Lipschitz, so there is nothing to test. In addition, in this case, not only the size of the range, but also the distances between points in the range play a role. Even though for monotonicity, repairing a function with a range of size greater than 2 in one dimension at a time does not work, this is exactly what we do here.

6.1.3 Organization

In Section 6.2, we present and analyze BubbleSmooth, our procedure for smoothing 1-dimensional functions, and prove Theorem 6.3. In Section 6.3, we use BubbleSmooth to construct a smoothing procedure for multidimensional functions that leads to the dimension reduction of Theorems 6.2 and 6.4. Our Lipschitz testers for functions on hypergrids claimed in Theorem 6.1 are presented in Section 6.4.

6.2 BubbleSmooth and its analysis

In this section, we describe BubbleSmooth and prove Theorem 6.3 which asserts that BubbleSmooth(f) outputs a Lipschitz function that does not differ too much from f in the $\ell_1$ distance. In Section 6.2.1, we present BubbleSmooth (Algorithm 3) and show that it outputs a Lipschitz function. Sections 6.2.2 and 6.2.3 are devoted to proving part (2) of Theorem 6.3. At the high level, the proof follows the ideas explained in Section 6.1.2 (right after Theorem 6.3). In Section 6.2.2, we define our transfer graph (Definition 6.3) and prove its key property (Lemma 6.1). In Section 6.2.3, we show that the existence of a matching with a large violation score implies that f is far from Lipschitz in the $\ell_1$ distance (Lemma 6.2) and complete the proof of part (2) of Theorem 6.3.
by constructing such a matching in the transfer graph.

6.2.1 Description of BubbleSmooth and proof of Part (1) of Theorem 6.3

We begin this section by recalling two basic definitions from Chapter 3.

Definition 6.1 (Violation score). Let $f$ be a function and $x, y$ be points in its domain. The pair $(x, y)$ is violated by $f$ if $|f(x) - f(y)| > |x - y|_1$. The violation score of $(x, y)$, denoted by $vs_f(x, y)$, is $|f(x) - f(y)| - |x - y|_1$ if it is violated and 0 otherwise.

Definition 6.2 (Basic operator). Given $f : [n]^d \to \mathbb{Z}$ and $x, y \in [n]^d$, where $|x - y|_1 = 1$ and vertex names $x$ and $y$ are chosen so that $f(x) \leq f(y)$, the basic operator $B_{x,y}$ works as follows: If the pair $(x, y)$ is not violated by $f$ then $B_{x,y}[f]$ is identical to $f$. Otherwise, $B_{x,y}[f](x) = f(x) + 1$ and $B_{x,y}[f](y) = f(y) - 1$.

In this section, we view a function $f : [n] \to \mathbb{Z}$ as an integer-valued sequence $f(1), f(2), \ldots, f(n)$. We denote the subsequence $f(i), f(i + 1), \ldots, f(j)$ by $f[i..j]$. Naturally, a sequence $f[i..j]$ is Lipschitz if $|f(k) - f(k + 1)| \leq 1$ for all $i \leq k \leq j - 1$.

Algorithm 3 presents a formal description of BubbleSmooth.

**Algorithm 3: BubbleSmooth (Input: an integer sequence $f[1..n]$)**

1. for $i = n - 1$ to 1 do
   1.1 // Start phase $i$. 
   2. while $|f(i) - f(i + 1)| > 1$ do // ($i, i+1$) is violated by $f$
      2.1 LinePass($i$).
3. return $f$

**Algorithm 4: LinePass (Input: integer $i$)**

1. for $j = i$ to $n - 1$ do
2. $f \leftarrow B_{i,j+1}[f]$. // Apply basic operator (see Definition 6.2.)

We start analyzing the behavior of BubbleSmooth by proving part (1) of Theorem 6.3, which states that BubbleSmooth returns a Lipschitz function.
Proof of part (1) of Theorem 6.3  Consider an integer sequence $f[1..n]$ and let $f'[1..n]$ be the sequence returned by BubbleSmooth$(f)$. We prove that $f'$ is Lipschitz by induction on the phase of BubbleSmooth. Initially, $f(n)$ is vacuously Lipschitz. We fix $i \in [n]$, assume $f[i+1..n]$ is Lipschitz at the beginning of phase $i$ and show this phase terminates and that $f[i..n]$ is Lipschitz at the end of the phase.

Consider an execution of LinePass$(i)$. Assume $f[i+1..n]$ is Lipschitz in the beginning of this execution. Let $j$ be the index, such that at the beginning of the execution, $f[i..j]$ is the longest strictly monotone sequence starting from $f(i)$. Then LinePass$(i)$ modifies two elements: $f(i)$ and $f(j)$. If $f(i) > f(j)$ then $f(i)$ is decreased by 1 and $f(j)$ is increased by 1, i.e., 1 unit is transferred from $i$ to $j$. Similarly, if $f(i) < f(j)$ then 1 unit is transferred from $j$ to $i$. It is easy to see that after this transfer is performed, $f[i+1..n]$ is still Lipschitz. Moreover, each iteration of LinePass$(i)$ reduces the violation score of the pair $(i, i+1)$ by at least 1. Thus, phase $i$ terminates with $f[i..n]$ being Lipschitz.

6.2.2 Transfer graph

In the proof of part (1) of Theorem 6.3, we established that for all $i \in [n]$, each iteration of LinePass$(i)$ transfers one unit to or from $i$. We record the transfers in the transfer graph $T = ([n], E)$, defined next. A transfer from $x$ to $y$ is recorded as a directed edge $(x, y)$. The edges of the transfer graph are ordered (indexed), according to when they were added to the graph. The edge $(i, j)$ (resp., $(j, i)$) corresponding to the most recent transfer is combined with a previously added edge $(j, k)$ (resp., $(k, j)$) if such an edge exists. This is done because transfers from $x$ to $y$ and from $y$ to $z$ are equivalent to a transfer from $x$ to $z$. If a new edge $(x, y)$ is merged with an existing edge $(y, z)$, the combined edge retains the index of the edge $(y, z)$.

Definition 6.3 (Transfer graph). The transfer graph $T = ([n], E)$, where the edge set $E = (e_1, \ldots, e_t)$ is ordered and edges are not necessarily distinct. The graph is defined by the following procedure. Initially, $E = \emptyset$ and $t = 0$. Each new run of LinePass during the execution of BubbleSmooth, transfers a unit from $i$ to $j$ (or resp., from $j$ to $i$) for some $i$ and $j$. If $j$ has no outgoing (resp., incoming) edge in $T$, then increment $t$ by 1 and add the edge $e_t = (i, j)$ (resp., $e_t = (j, i)$) to $E$. Otherwise, let $e_s$ be an outgoing edge $(j, k)$ (resp., an incoming edge $(k, j)$) with the largest index $s$. Replace $(j, k)$ with
(i, k), i.e., \( e_s \leftarrow (i, k) \). \( \text{Replace} \ (k, j) \ \text{with} \ (k, i), \ i.e., \ e_s \leftarrow (k, i). \) \text{The final transfer graph is denoted by} \ T^*. \)

As mentioned previously, the order of creation of edges is important to formalize the desired property of the transfer graph, so we need to consider the subgraphs that consist of the first \( s \) edges \( e_1, \ldots, e_s \) of \( E \).

**Definition 6.4** (Degrees). Consider a transfer graph \( T \) at some time during the execution of \textit{BubbleSmooth}. For all \( s \in \{0, \ldots, t\} \) its subgraph graph \( T_s \) is defined as \( ([n], (e_1, \ldots, e_s)) \), where \( (e_1, \ldots, e_t) \) is the ordered edge set of \( T \). (When \( s = 0 \), the edge set of \( T_s \) is empty.) The degree of a vertex \( x \in [n] \) of \( T_s \) is denoted by \( \deg_s(x) \); when \( T_s \) is a subgraph of the final transfer graph, it is denoted by \( \deg^*_s(x) \).

Observe that at no point in time can a vertex in \( T \) simultaneously have an incoming and an outgoing edge because such edges would get merged into one edge.

**Lemma 6.1** (Key property of transfer graph). Let \( f \) be an input function given to \textit{BubbleSmooth}. Then for each edge \( e_s = (x, y) \) of the final transfer graph \( T^* \), the following holds: \( \forall s \), \( f(x) - f(y) \geq \deg_s(x) + \deg_s(y) - 1 \).

To prove this lemma, we consider each phase of \textit{BubbleSmooth} separately and formulate a slightly stronger invariant that holds at every point during that phase.

**Definition 6.5.** For all \( i \in [n - 1] \), let \( \Delta_i \) be the degree of \( i \) in the transfer graph at the end of phase \( i \).

The following stronger invariant of the transfer graph directly implies Lemma 6.1.

**Claim 6.1** (Invariant for phase \( i \)). Let \( f \) be an input function given to \textit{BubbleSmooth}. At every point during the execution of \textit{BubbleSmooth}(\( f \)), for each edge \( e_s = (x, y) \) of the transfer graph \( T \),

\[
f(x) - f(y) \geq \deg_s(x) + \deg_s(y) - 1 + |x - y|.
\]

Moreover, for each phase \( i \in [n - 1] \), after each execution of \textit{LinePass}(\( i \)), for each edge \( e_s \) incident on vertex \( i \), the following (stronger) condition holds:

\[
\text{if the edge } e_s = (i, j), \text{ i.e., it is outgoing from } i, \text{ then } f(i) - f(j) \geq \Delta_i + \deg_s(j) - 1 + |i - j|;
\]
if the edge $e_s = (j, i)$, i.e., it is incoming into $i$, then $f(j) - f(i) \geq \Delta_i + \text{deg}_s(j) - 1 + |i - j|$.

Observe that all transfers involving $i$ during phase $i$ are in the same direction: if in the beginning of the phase we have $f(i) > f(i + 1)$, then all transfers are from $i$; if we have $f(i) < f(i + 1)$ instead, then all transfers are to $i$. In particular, whenever an edge incident to $i$ is added, it is not modified subsequently during phase $i$. So for all $s$, $\text{deg}_s(i)$ never exceeds $\Delta_i$ during phase $i$ and the condition in Claim 6.1 is indeed stronger than that in Lemma 6.1.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.1.png}
\caption{Two cases in the proof of Claim 6.1.}
\end{figure}

**Proof of Claim 6.1** Initially the transfer graph has no edges, so the invariant stated in Claim 6.1 holds. Observe that for all $i \in [n - 2]$, if the invariant holds at the end of phase $i + 1$, it also holds in the beginning of the following phase $i$. This is because the condition on each edge not incident to $i$ in phase $i + 1$ is the same or stronger than in phase $i$ (notice that in the beginning of phase $i$ there are no edges incident to $i$). It remains to prove that if the invariant holds before an iteration of LinePass, it also holds after the iteration.

Consider a phase $i \in [n - 1]$ and an execution of LinePass($i$) that transfers a unit from $i$ to $j$ for some $j \in \{i + 1, \ldots, n\}$. (The case when a unit is transferred in the other direction is symmetric.) Let $f^-$ be the function and $T^-$ be the transfer graph right before the considered execution of LinePass. Define $\text{deg}^-_s$ as in Definition 6.4, with respect to the transfer graph $T^-$. Define $T^+$ and $\text{deg}^+_s$ analogously for the moment right after the considered execution of LinePass. Let $t$ be the number of edges in $T^-$. Since the current transfer occurred, the sequence $f^-[i+1, j]$ is monotone decreasing, giving $f^-(i + 1) - f^-(j) \geq |i + 1 - j|$. The number of transfers from $i$ that occurred before the current transfer is $\text{deg}^-_i(i)$. The number of the remaining transfers, including the current one, in phase $i$ is thus $\Delta_i - \text{deg}^-_i(i)$. Since each such transfer from $i$ can happen only if the pair $(i, i + 1)$ is violated, and moreover it lowers the violation score.
of the pair \((i, i + 1)\) by at least 1, it follows that \(v_{f^{-}}(i, i + 1) \geq \Delta_i - \deg^-(i)\) or, equivalently, \(f^{-}(i) - f^{-}(i + 1) \geq \Delta_i - \deg^-(i) + 1\). Therefore,

\[
f^{-}(i) - f^{-}(j) = [f^{-}(i) - f^{-}(i+1)] + [f^{-}(i+1) - f^{-}(j)] \geq \Delta_i - \deg^-(i) + |i - j|. \tag{6.1}
\]

The effect the current transfer has on the transfer graph \(T^{-}\) depends on whether \(T^{-}\) contains an outgoing edge from \(j\). We consider the two corresponding cases separately.

**Case 1:** transfer graph \(T^{-}\) contains no outgoing edge from \(j\). Then \(T^{+}\) is obtained from \(T^{-}\) by adding the edge \(e_{t+1} = (i, j)\). (See Figure 6.1.)

Recall that all transfers involving \(i\) made during phase \(i\) are in the same direction, either all from \(i\) or all to \(i\). So, by the assumption that the current transfer is from \(i\), vertex \(i\) can have only outgoing edges in the transfer graph during phase \(i\). That is,

\[
f(i) = f^{-}(i) + \deg^-(i). \tag{6.2}
\]

By assumption of Case 1, vertex \(j\) can have only incoming edges. That is,

\[
f(j) = f^{-}(j) - \deg^-(j). \tag{6.3}
\]

Applying first (6.2), (6.3), and then (6.1), we get:

\[
\begin{align*}
f(i) - f(j) & = f^{-}(i) - f^{-}(j) + \deg^-(i) + \deg^-(j) \\
& \geq \Delta_i - \deg^-(i) + |i - j| + \deg^-(i) + \deg^-(j) \\
& = \Delta_i + \deg^+_t(j) - 1 + |i - j|.
\end{align*}
\]

The last equality holds because the edge \(e_{t+1} = (i, j)\) is added to \(T\) after the current transfer, so \(\deg^+_t(j) = \deg^-(j)+1\). We proved that the invariant in Claim 6.1 holds for the new edge.

Since all other edges and their ordering are unchanged, \(\deg^+_s(x) = \deg^-_s(x)\) for all \(x \in [n]\) and \(s \leq t\). Thus, the invariant of Claim 6.1 holds for all edges of \(T^{+}\).

**Case 2:** transfer graph \(T^{-}\) contains an outgoing edge from \(j\). Let \(e_r = (j, k)\) be such an edge with the largest index \(r\). Then \(T^{+}\) is obtained from \(T^{-}\) by replacing the edge
with the edge \((i, k)\), with this new edge receiving the index \(r\). (See Figure 6.1.) Notice that \(k\) could be larger or smaller than \(j\).

Recall that each vertex in \(T^-\) can only have one type of incident edges: either incoming or outgoing. In this case, both \(i\) and \(j\) can only have outgoing edges. Since \(T^-\) has \(t\) edges,

\[
f(i) = f^-(i) + \text{deg}_r^-(i) \quad \text{and} \quad f(j) = f^-(j) + \text{deg}_r^-(j). \tag{6.4}
\]

By assumption, the invariant in Claim 6.1 holds for the transfer graph \(T^-\). Since the edge \((j, k)\) has index \(r\) in \(T^-\), i.e., \((j, k) = e_r^r\),

\[
f(j) - f(k) \geq \text{deg}_r^-(j) + \text{deg}_r^-(k) - 1 + |j - k|.
\]

Finally, \(\text{deg}_r^-(j) = \text{deg}_r^-(j)\) because of our choice of \(r\). Putting the equations for this case together, then applying (6.1) and finally using the triangle inequality, we get:

\[
f(i) - f(k) = [f(i) - f(j)] + [f(j) - f(k)]
\geq [f^-(i) + \text{deg}_r^-(i) - f^-(j) - \text{deg}_r^-(j)]
+ [\text{deg}_r^-(j) + \text{deg}_r^-(k) - 1 + |j - k|]
\geq \Delta_i - \text{deg}_r^-(i) + |i - j| + \text{deg}_r^-(i) + \text{deg}_r^-(k) - 1 + |j - k|
\geq \Delta_i + \text{deg}_r^-(k) - 1 + |i - k| = \Delta_i + \text{deg}_r^+(k) - 1 + |i - k|.
\]

Thus, the invariant in Claim 6.1 holds for the newly formed edge.

When the transfer graph is modified to reflect the current transfer, only edges incident to \(i, j\) or \(k\) are affected by the changes. For all other nodes \(x\) and all \(s < t\), the corresponding degrees remain the same: \(\text{deg}^+_s(x) = \text{deg}^-_s(x)\). For the node \(k\) and all \(s < t\), the degrees are unchanged: \(\text{deg}^+_s(k) = \text{deg}^-_s(k)\). For the node \(j\) and all \(s \leq t\), the degrees decrease or remain the same: \(\text{deg}^+_s(j) \leq \text{deg}^-_s(j)\). Therefore, for all edges in \(T^+\) not affected by the current merge and not incident to \(i\), the invariant in Claim 6.1 still holds. Finally, the stronger invariant for edges incident to \(i\) also holds for \(T^+\) because it depends on a fixed parameter \(\Delta_i\) (instead of \(\text{deg}^-_s(i)\)).
6.2.3 Matchings of violated pairs

Part (2) of Theorem 6.3 states that the $\ell_1$ distance between $f$ and $\text{BubbleSmooth}(f)$ is at most $2 \cdot \ell_1^{\text{Lip}}(f)$. By definition of the transfer graph $T = ([n], E)$, the distance $|f - \text{BubbleSmooth}(f)|_1 = 2|E|$. Lemma 6.2 shows that $\ell_1^{\text{Lip}}(f)$ is bounded below by the violation score of any matching. We complete the proof of Theorem 6.3 by showing that $T$ has a matching with violation score $|E|$.

**Definition 6.6 (Violation score of a set of pairs).** Let $M$ be a set of pairs of pairs violated by $f$. The violation score of the set $M$, denoted $VS(M)$, is the sum of violation scores of all pairs in $M$.

**Lemma 6.2.** Let $M$ be a matching of pairs $(x,y)$ where $x$ and $y$ are in the (discrete) domain of a function $f$. Then $\ell_1^{\text{Lip}}(f) \geq VS(M)$.

**Proof.** Let $f^*$ be a closest Lipschitz function to $f$ (on the same domain as $f$) with respect to the $\ell_1$ distance, i.e., $|f - f^*|_1 = \ell_1(f, \text{Lip})$. Consider a pair $(x,y) \in M$. Since $|f(x) - f(y)| = d(x,y) + vs_f(x,y)$ and $|f^*(x) - f^*(y)| \leq d(x,y)$, it follows by the triangle inequality that $|f(x) - f^*(x)| + |f(y) - f^*(y)| \geq vs_f(x,y)$. Since $M$ is a matching, we can add over all of its pairs to obtain

$$\ell_1(f, \text{Lip}) = |f - f^*|_1 \geq \sum_{(x,y) \in M} (|f(x) - f^*(x)| + |f(y) - f^*(y)|) \geq \sum_{(x,y) \in M} vs_f(x,y) = vs_f(M),$$

which concludes the proof. \hfill $\square$

Now using Lemma 6.1 we exhibit a matching in the final transfer graph which has large violation score, concluding the proof of Theorem 6.3.

**Proof of part (2) of Theorem 6.3** Let $T^* = ([n], E)$ be the final transfer graph corresponding to the execution of $\text{BubbleSmooth}$ on $f$ and let $E = \{e_1, \ldots, e_t\}$. By definition of the transfer graph, $|f - f'|_1 = \sum_{i \in [n]} \text{deg}_t(i) = 2|E|$. By Lemma 6.2, it is enough to show that there is a matching $M$ of pairs violated by $f$ with the violation score $vs_f(M) \geq |E|$.
We claim that $T$ contains such a matching. It can be constructed greedily by repeating the following step, starting with $s = t$: add $e_s$ to $M$ and then remove $e_s$ and all other edges adjacent to its endpoints from $T$; set $s$ to be the number of edges remaining in $E$. In each step, at most $\deg s(x) + \deg s(y) - 1$ are removed from $T$. (“At most” because $T$ can have multiple edges.) By Lemma 6.1, $v_{s_f}(x, y) \geq \deg s(x) + \deg s(y) - 1$. So, at each step of the greedy procedure, the violation score of the pair $(x, y)$ added to $M$ is at least the number of edges removed from $T$. Therefore, $v_{s_f}(M) \geq |E|$. 

6.3 Dimension reduction: proof of Theorems 6.2 and 6.4

In this section, we prove Theorems 6.2 and 6.4 that connect the distance of a function to being Lipschitz to the distance of its lines to being Lipschitz. Effectively, these results reduce the task of testing a multidimensional function to the task of testing its lines. Our main contribution in this section is a smoothing procedure that makes a function Lipschitz by modifying it at a few points by repairing one dimension at a time. In Section 6.3.1, we present the dimension operator that repairs all lines in a specified dimension by applying BubbleSmooth to each of them. The important properties of the dimension operator are summarized in Lemma 6.3 which is the key ingredient in the proofs of Theorems 6.2 and 6.4. Section 6.3.2 proves auxiliary claims used in the proof of Lemma 6.3. Section 6.3.3 completes the proofs of Theorems 6.2 and 6.4.

6.3.1 Dimension operator and its properties

Recall from the discussion in Section 6.1.2 that we denote the set of lines of $f$ along dimension $i$ by $L^i_f$ and the set of all lines of $f$ by $L_f = L^i_f$.

**Definition 6.7** (Dimension operator $A_i$). Given $f : [n]^d \rightarrow \mathbb{Z}$ and dimension $i \in [d]$, the dimension operator $A_i$ applies BubbleSmooth to every function $g \in L^i_f$ and returns the resulting function.

Next lemma summarizes the properties of the dimension operator.

**Lemma 6.3** (Properties of the dimension operator $A_i$). For all $i \in [d]$, the dimension operator $A_i$ satisfies the following properties for every function $f : [n]^d \rightarrow \mathbb{Z}$.

1. (Repairs dimension $i$.) Every $g \in L^i_{A_i[f]}$ is Lipschitz.
2. (Does not modify the function too much.) \(|f - A_i[f]|_1 \leq 2 \cdot \sum_{g \in L^i_j} \ell_1^{Lip}(g)\).

3. (Does not spoil other dimensions.) For all \(j \neq i\) in \([d]\), it does not increase the expected \(\ell_1\) distance of a random line in dimension \(j\) to the Lipschitz property, i.e.,
\[
\mathbb{E}_{g \leftarrow L^j_{\alpha_i}[f]}[\ell_1^{Lip}(g)] \leq \mathbb{E}_{g \leftarrow L^j_{\alpha_i}[f]}[\ell_1^{Lip}(g)].
\]

**Proof.** Item 1. Item 1 follows from part (1) of Theorem 6.3.

**Item 2.** Since the dimension operator \(A_i\) operates by applying \texttt{BubbleSmooth} to all (disjoint) lines in \(L^i_j\), we get \(|f - A_i[f]|_1 = \sum_{g \in L^i_j}|g - \texttt{BubbleSmooth}[g]|_1\). The latter is at most \(\sum_{g \in L^i_j} 2 \cdot \ell_1^{Lip}(g)\) by Part (2) of Theorem 6.3, thus proving the item.

**Item 3.** Fix \(i\) and \(j\). First, we give a standard argument from [GGL+00, DGL+99] (and also used in Chapters 3 and 4) that it is enough to prove this statement for \(n \times n\) grids. Namely, every \(\alpha \in [n]^d\) with \(\alpha_i = \alpha_j = 0\) defines a restriction of a function \(f\) to an \(n \times n\) grid by \(h(x_i, x_j) = f(\alpha + x_i \cdot e^i + x_j \cdot e^j)\), where \(x_i\) and \(x_j\) range over \([n]\). (Recall that \(e^i \in [n]^d\) is 1 on the \(i\)th coordinate and 0 on the remaining coordinates.) If the item holds for all 2-dimensional grids, we can average over all such grids defined by different \(\alpha\) to obtain the statement for the \(d\)-dimensional function \(f\). Now fix an arbitrary restriction \(h : [n]^2 \rightarrow \mathbb{Z}\) as discussed and think of \(h\) as an \(n \times n\) matrix with rows (resp., columns) corresponding to lines in dimension \(i\) (resp., in dimension \(j\)).

The key feature of our dimension operator \(A_i\) is that it can be broken down into steps, each consisting of one application of the basic step of \texttt{BubbleSmooth} to the same positions \((k, k + 1)\) on all lines in dimension \(i\). To see this, observe that we can replace the while loop condition on Line 2 of Algorithm 4 with “repeat \(t\) times”, where \(t\) should be large enough to guarantee that the line segment under consideration is Lipschitz after \(t\) iterations of \texttt{LinePass}. (E.g., \(t = n \cdot \text{ImD}(f)\) repetitions suffices.) If this version of \texttt{BubbleSmooth} is run synchronously and in parallel on all lines in dimension \(i\), the basic step will be applied to the same positions \((k, k + 1)\) on all lines.

Since in each parallel update step only two adjacent columns of \(h\) are affected, it is sufficient to prove the item for two adjacent columns of \(h\). Accordingly, consider two adjacent columns \(C_1\) and \(C_2\) of \(h\). Let \(M_1\) and \(M_2\) be Lipschitz columns that are closest in the \(\ell_1\) distance to \(C_1\) and \(C_2\), respectively. Thus, \(\ell_1^{Lip}(C_1) = |C_1 - M_1|_1\) and \(\ell_1^{Lip}(C_2) = |C_2 - M_2|_1\). Let \(C'_1\) and \(C'_2\) be the columns of the matrix resulting from applying the basic operator to the rows of the matrix \((C_1, C_2)\). Similarly, define \(M'_1\) and \(M'_2\) to be the columns of the matrix resulting from applying the basic operator to the
rows of \((M_1, M_2)\). By Corollary 6.1, \(M'_1\) and \(M'_2\) are Lipschitz. Therefore, \(\ell_{1\text{Lip}}^L(C'_1) \leq \left|C'_1 - M'_1\right|_1\) and \(\ell_{1\text{Lip}}^L(C'_2) \leq \left|C'_2 - M'_2\right|_1\). Finally, using the inequality \(\left|C'_1 - M'_1\right|_1 + \left|C'_2 - M'_2\right|_1 \leq |C_1 - M_1|_1 + |C_2 - M_2|_1\) proved in Corollary 6.2 below, the proof of Item 3 is completed as follows:

\[
\ell_{1\text{Lip}}^L(C_1) + \ell_{1\text{Lip}}^L(C_2) = |C_1 - M_1|_1 + |C_2 - M_2|_1 \\ \geq |C'_1 - M'_1|_1 + |C'_2 - M'_2|_1 \geq \ell_{1\text{Lip}}^L(C'_1) + \ell_{1\text{Lip}}^L(C'_2).
\]

\[\square\]

### 6.3.2 Basic operator on a square

To analyze the behavior of the dimension operator, we need to understand the effect of the basic operator \(B_{k,k+1}\) on a multidimensional function. We remark that our definition of the basic operator coincides with definition of basic operator of Chapter 3 (see Definition 6.2) for integer-valued functions defined on domain \(\{0,1\}^2\). We recall and extend properties of the basic operator from Chapter 3 when applied to functions defined on \(\{0,1\}^2\) in Claim 6.2 below.

**Claim 6.2** (Properties of basic operator on a square). Consider an integer-valued function \(f\) defined on domain \(\{x_t, x_b, y_t, y_b\}\), where \(x_t, x_b, y_t\) and \(y_b\) denote 10, 00, 11 and 01, respectively. (See Figure 6.2.) Let \(f'\) be the function obtained by applying basic operators \(B_{x_t, y_t}\) and \(B_{x_b, y_b}\) along the horizontal edges. Then the following holds for the vertical edges.

1. **Violation score of vertical edges did not increase**: \(\mathit{vS}_{f'}(x_t, x_b) + \mathit{vS}_{f'}(y_t, y_b) \leq \mathit{vS}_f(x_t, x_b) + \mathit{vS}_f(y_t, y_b)\). (See proof of Lemma 3.3 of Chapter 3.)

2. **The absolute difference of values along vertical edges did not increase**: \(\left|f'(x_t) - f'(x_b)\right| + \left|f'(y_t) - f'(y_b)\right| \leq \left|f(x_t) - f(x_b)\right| + \left|f(y_t) - f(y_b)\right|\).

**Proof of Claim 6.2, Item 2.** If neither horizontal edge is violated then \(f' = f\) and we are done. So assume w.l.o.g. \(\{x_t, y_t\}\) is violated such that \(f(y_t) \geq f(x_t) + 2\). Then \(B_{x_t, y_t}\) increases \(f(x_t)\) by 1 and decreases \(f(y_t)\) by 1 leading to Inequality (i): \(f'(y_t) \geq f(x_t) + 1\). If the absolute difference on the value of both vertical edges do not increase, we are
also done. So assume w.l.o.g. that the absolute difference of the values of the left vertical edge increases strictly, namely \(|f'(x_b) - f'(x_t)| = |f(x_b) - f(x_t)| + \Delta\) for \(\Delta \in \{1, 2\}\). This implies that \(B_{x_b, y_b}\) did not increase the value at \(x_b\) and the following holds: (ii) \(f(x_b) \leq f(x_t)\). The former also implies that: (iii) \(f'(y_b) \leq f'(x_b) + 1\). The definition of \(\Delta\) further gives that: (iv) \(f'(x_b) = f(x_b) - (\Delta - 1)\) and (v) \(f'(y_b) = f(y_b) + (\Delta - 1)\).

Using inequalities (i), (ii), (iv) and (iii) we get

\[
f'(y_t) \geq f(x_t) + 1 \geq f(x_b) + 1 = f'(x_b) + \Delta \geq f'(y_b) + (\Delta - 1),
\]

and hence \(f'(y_t) - f'(y_b) \geq 0\). Using the last inequality, equation (v) and the fact that \(f'(y_t) = f(y_t) - 1\), we get

\[
|f'(y_t) - f'(y_b)| = f'(y_t) - f'(y_b) = f'(y_t) - f(y_b) - (\Delta - 1) = f(y_t) - f(y_b) - \Delta \leq |f(y_t) - f(y_b)| - \Delta.
\]

Thus, the absolute difference of the values for \(\{x_t, x_b\}\) increases by \(\Delta\), while the same for \(\{y_t, y_b\}\) decreases by \(\Delta\), proving the claim.

We now use Claim 6.2 to prove Corollaries 6.1 and 6.2 which were used in the proof of Lemma 6.3.

**Corollary 6.1.** Let \(M\) be a matrix consisting of two Lipschitz columns. If the basic operator is applied to the rows of this matrix then the resulting matrix \(M'\) still has Lipschitz columns.
Proof. Applying part 1 of Claim 6.2 to each $2 \times 2$ grid formed by taking 2 adjacent rows of $M$ (respectively, $M'$), we get the desired statement.

The second corollary is about one-dimensional functions

$$C_1, C_2, M_1, M_2, C'_1, C'_2, M'_1 \text{ and } M'_2$$

used in proof of Lemma 6.3.

**Corollary 6.2.** $|C_1 - M_1|_1 + |C_2 - M_2|_1 \geq |C'_1 - M'_1|_1 + |C'_2 - M'_2|_1$.

Proof. For every $z \in [n]$, we show that the following holds. This implies the corollary by summing over all $z \in [n]$.

$$|(C_1(z) - M_1(z)) + (C_2(z) - M_2(z))| - (|C'_1(z) - M'_1(z)| + |C'_2(z) - M'_2(z)|) \geq 0.$$

Fix $z \in [n]$ and let $f : \{0, 1\}^2 \to \mathbb{Z}$ be the function defined as follows: $f(x_b) = C_1(z)$, $f(y_b) = M_1(z)$, $f(x_t) = C_2(z)$ and $f(y_t) = M_2(z)$. Similarly, let $f' : \{0, 1\}^2 \to \mathbb{Z}$ be defined as follows: $f'(x_b) = C'_1(z)$, $f'(y_b) = M'_1(z)$, $f'(x_t) = C'_2(z)$ and $f'(y_t) = M'_2(z)$. Then the above inequality follows from the second part of Claim 6.2. Hence the corollary follows.

6.3.3 Proof of Theorems 6.2 and 6.4

To prove Theorems 6.2 and 6.4, we use the following observation that relates $\ell_1^{Lip}(f)$ to $\varepsilon^{Lip}(f)$.

**Observation 6.1.** For all $f : [n]^d \to \delta\mathbb{Z}$, the following holds: $\delta\varepsilon^{Lip}(f) \cdot n^d \leq \ell_1^{Lip}(f) \leq \varepsilon^{Lip}(f) \cdot n^d \cdot \text{ImD}(f)$.

The first inequality follows directly from definitions, while the second follows from the fact that one can make a function $f$ Lipschitz by changing $\varepsilon^{Lip}(f)$ fraction of values, each by at most $\text{ImD}(f)$.

Proof of Theorems 6.2 and 6.4 Let $A_i$ be the dimension repair operator of Definition 6.7. For $i \in [d]$, define $f_i$ inductively by letting $f_i = A_i[f_{i-1}]$ with the base case being $f_0 = f$. Items 1 and 3 of Lemma 6.3 give that $f_d$ is Lipschitz. Specifically, Item
1 implies that the application of the dimension operator $A_i$ makes $f_{i-1}$ Lipschitz along the $i$th dimension while Item 3 ensures that each such application does not introduce violations in the already repaired dimensions. Using properties of $A_i$ from Lemma 6.3, the following holds for all $i \in [d]$.

$$|f_{i-1} - f_i|_1 = |f_{i-1} - A_i[f_{i-1}]|_1 \leq 2 \cdot \sum_{g \in L_{f_{i-1}}} \ell_1^{\text{Lip}}(g) \leq 2 \cdot \sum_{g \in L_f} \ell_1^{\text{Lip}}(g).$$

Specifically, the two inequalities above follow from Items 2 and 3 of Lemma 6.3, respectively. By the triangle inequality, $\ell_1^{\text{Lip}}(f) \leq \sum_{i=1}^{d} |f_{i-1} - f_i|_1$. This fact together with the above bound on $|f_{i-1} - f_i|_1$, leads to the following chain of (in)equalities and proves Theorem 6.4.

$$\ell_1^{\text{Lip}}(f) \leq \sum_{i=1}^{d} |f_{i-1} - f_i|_1 \leq \sum_{i=1}^{d} 2 \cdot \sum_{g \in L_f} \ell_1^{\text{Lip}}(g) = 2 \cdot \sum_{g \in L_f} \ell_1^{\text{Lip}}(g).$$

For proving Theorem 6.2, we apply Observation 6.1 to both sides of the inequality of Theorem 6.4 leading to the first inequality below.

$$n^d \epsilon^{\text{Lip}}(f) \leq 2 \cdot \sum_{g \in L_f} \epsilon^{\text{Lip}}(g) \cdot \text{ImD}(g) \cdot n \leq 2 \cdot \text{ImD}(f) \cdot n \sum_{g \in L_f} \epsilon^{\text{Lip}}(g) = 2 \cdot d \cdot \text{ImD}(f) \cdot n^d \cdot \mathbb{E}_{g \in L_f}[\epsilon^{\text{Lip}}(g)].$$

The last inequality follows from the fact that $\text{ImD}(f)$ is a trivial upper bound on $\text{ImD}(g)$ for every $g \in L_f$. Finally, expressing the sum as an expectation (as done in the last equality), we get Theorem 6.2.

6.4 Testing the Lipschitz property on hypergrids

In this section, we present our testers for the Lipschitz property of functions $f : [n]^d \to \mathbb{Z}$. Theorem 6.2 relates the distance of a function $f$ from the Lipschitz property to the (expected) distance of its lines to this property. The resulting bound, however, depends on the image diameter of $f$. The image diameter is small (at most $nd$) for Lipschitz
functions, but can be arbitrarily large otherwise. The high-level description of our testers is the following: (i) estimate the image diameter of $f$ and reject if it is too large; (ii) repeatedly sample a line $g$ of $f$ at random, run one step of a Lipschitz tester for the line on $g$ and reject if a violated pair is discovered; otherwise, accept. Step (i) ensures that a small sample of lines is enough to succeed with constant probability. The testers differ only in one parameter which quantifies what “too large” means in Step (i).

6.4.1 Estimating the effective image diameter

As mentioned before, a Lipschitz function on $[n]^d$ has image diameter at most $nd$, which can serve as a threshold for rejection in Step (i) of the informal procedure above. However (if we are willing to tolerate two-sided error), it is sufficient to use a smaller threshold, equal the effective diameter of the function. For a given $\epsilon \in (0, 1]$, define $\text{ImD}_\epsilon(f)$ as the smallest value $\alpha$ such that $f$ is $\epsilon$-close to having image diameter $\alpha$:

$$\text{ImD}_\epsilon(f) = \min_{U \subseteq [n]^d : |U| \geq (1-\epsilon)nd} \{ \max_{x \in U} f(x) - \min_{x \in U} f(x) \}.$$ 

Although the image diameter of a Lipschitz function $f$ can indeed achieve value $nd$, the effective $\text{ImD}_\epsilon(f)$ is upper bounded by the potentially smaller quantity $O(n\sqrt{d\ln(1/\epsilon)})$.

The next lemma makes this precise, and follows directly from the well-known McDiarmid’s inequality. We state McDiarmid’s inequality [McD89b] specialized to the domain $[n]^d$.

**Theorem 6.5** ([McD89b]). For every Lipschitz function $f : [n]^d \to \mathbb{R}$ and uniformly distributed $X \in [n]^d$, the following holds (where the expectation is taken over the uniformly distributed $X$): $\Pr[|f(X) - \mathbb{E}[f(X)]| \geq t] \leq 2e^{-\frac{2t^2}{dn^2}}$.

**Lemma 6.4** (Effective image diameter of Lipschitz functions). For all $\epsilon \in (0, 1]$, each Lipschitz function $f : [n]^d \to \mathbb{R}$ is $(\epsilon/21)$-close to having image diameter at most $n\sqrt{d\ln(42/\epsilon)}$.

Our testers use estimates of image diameter or effective diameter to reject functions. The next lemma shows that we can get such estimates efficiently. An algorithm satisfying parts (i) and (ii) of the lemma was stated and proved in Claim 3.1 of Chapter 3.
Lemma 6.5. There is a randomized algorithm \textsc{Sample-Diameter} that, given a function \( f : [n]^d \rightarrow \mathbb{R} \) and \( \epsilon \in (0, 1] \), outputs an estimate \( r \in \mathbb{R} \) such that: (i) \( \text{ImD}_\epsilon(f) \leq r \) with probability at least \( 5/6 \); (ii) \( r \leq \text{ImD}(f) \) (always) and (iii) \( r \leq \text{ImD}_{\epsilon/21}(f) \) with probability at least \( 2/3 \). Moreover, the algorithm runs in time \( O(1/\epsilon) \).

Proof. \textsc{Sample-Diameter} is the same algorithm as stated in Claim 3.1 of Chapter 3: sample \( s = \lceil 6/\epsilon \rceil \) points \( z_1, \ldots, z_s \in [n]^d \) and output \( r = \max_{z_i} f(z_i) - \min_{z_i} f(z_i) \).

The first two parts of the lemma were stated in Claim 3.1 and proved in Chapter 3. Therefore, it remains to show that with probability at least \( 2/3 \), \( r \leq \text{ImD}_{\epsilon/21}(f) \).

By the definition of \( \text{ImD}_{\epsilon/21}(f) \), let \( S \subseteq [n]^d \) be a set of size at most \( \epsilon n^d / 21 \) such that for every \( x, y \in [n]^d \setminus S \) we have \( |f(x) - f(y)| \leq \text{ImD}_{\epsilon/21}(f) \). We have \( r \leq \text{ImD}_{\epsilon/21}(f) \) whenever all the samples \( z_i \)'s lie outside \( S \); by the union bound, the probability of this event is at least \( 1 - \left( \frac{6}{\epsilon} + 1 \right) \left( \frac{21}{21} \right) \geq 2/3 \), as required.

6.4.2 Tester for hypergrid domains

Our tester for functions on hypergrids uses a tester for functions on lines from Chapter 5. In particular, we use the following lemma which follows immediately from Claim 5.1 of Chapter 5.

Lemma 6.6. Consider a function \( g : [n] \rightarrow \mathbb{R} \) and \( r \geq \text{ImD}(g) \). Then there is a 1-sided error algorithm \textsc{Line-Tester} which on input \( g \) and \( r \) rejects with probability at least \( \frac{1}{10 \log \min\{r,n\}} \).

To analyze our testers, we also need to estimate the probability that a random line \( g \leftarrow L_f \) is rejected by \textsc{Line-Tester}(\( g, r \)) with \( r \geq \text{ImD}_{\epsilon/2}(f) \). Such bound \( r \) will be obtained via Lemma 6.5. Since \( r \) may be much smaller than \( \text{ImD}(f) \), Lemma 6.6 does not apply directly. Nevertheless, the next lemma shows how to circumvent this difficulty.

Lemma 6.7. Let \( f : [n]^d \rightarrow \mathbb{Z} \) be \( \epsilon \)-far from Lipschitz. Consider a real \( r \geq \text{ImD}_{\epsilon/2}(f) \). For a random line \( g \leftarrow L_f \), the probability that \textsc{Line-Tester}(\( g, r \)) rejects is at least \( \frac{\epsilon}{40dr \log \min\{r,n\}} \).

Proof. Define the function \( f' \) by truncating \( f \) as follows. Using the definition, consider integers \( a < b \) such \( |a - b| \leq \text{ImD}_{\epsilon/2}(f) \) and at most \( \epsilon n^d / 2 \) points \( x \in [n]^d \) have
construction of $f(x)$ must be also violated with respect to $g$. To see this, notice that actually every pair $x, y$ is also violated for $g$ obtained by setting $a$ value in the construction of $f$. Clearly $ImD(f') \leq ImD_{\epsilon/2}(f) \leq r$ and, since $f$ and $f'$ differ on at most $\epsilon/2$ points, $f'$ is $\epsilon/2$-far from Lipschitz.

We first analyze how LINE-TESTER behaves for $f'$. Let $g' \leftarrow L_{f'}$ be a random line of $f'$. Since now $r$ is an upper bound on $ImD(f')$, and hence $r$ is also an upper bound on $ImD(g')$, we can use Lemma 6.6 to obtain that LINE-TESTER($g', r$) rejects with probability at least $\mathbb{E}_{g' \leftarrow L_{f'}}(\epsilon^{Lip}(g')/10 \log \min\{r, n\})$. Now we can use the dimension reduction Theorem 6.2 to lower bound this probability of rejection by

$$\frac{\epsilon^{Lip}(f')}{20d ImD(f') \log \min \{r, n\}} \geq \frac{\epsilon}{40d r \log \min \{r, n\}}.$$  

We claim the following, which directly implies the desired result: consider a line $g \in L_f$ of function $f$ and let $g'$ be the corresponding line of $f'$; the probability that LINE-TESTER($g, r$) rejects is at least the probability that LINE-TESTER($g', r$) rejects. To see this, notice that actually every pair $x, y \in [n]$ that is violated with respect to $g'$, must be also violated with respect to $g$: Assume without loss of generality that $g'(x) < g'(y) - |x - y|$. We obtain in particular that $g'(x) < \max_z f'(z) \leq b$. Notice that by construction of $f'$, this also implies that the $g(x) \leq g'(x)$, since whenever we decrease a value in the construction of $f'$ from $f$ this new value becomes equal to $b$. Similarly, $g'(y) > a$ and hence $g(y) \geq g'(y)$. These bonds give that $g(x) < g(y) - |x - y|$ and hence $(x, y)$ is violated with respect to $g$. Since every pairs violated for $g'$ is also violated for $g$, the fact that LINE-TESTER only test pairs for violations implies the claim. This also concludes the proof of the lemma.

Algorithm 5 presents our tester for the Lipschitz property on hypergrid domains. One of its inputs is a threshold $R$ for rejection in Step 1. The testers in Theorem 6.1 are obtained by setting $R$ appropriately.

**Algorithm 5:** Tester for Lipschitz property on hypergrid.

input : function $f : [n]^d \rightarrow \mathbb{Z}$, $\epsilon \in (0, 1]$, and value $R \in \mathbb{R}$

1. Let $r \leftarrow \text{SAMPLE-DIAMETER}(f, \epsilon/2)$. If $r > R$, reject.
2. for $i = 1$ to $\ell = \frac{80d r \log \min\{r, n\}}{\epsilon}$ do
   3. Select a line $g$ uniformly from $L_f$ and reject if LINE-TESTER($g, r$) does.
4. Accept.
Proof of Theorem 6.1 We show that Algorithm 5 when run with $R = nd$ (respectively, $R = n\sqrt{d\ln(84/\epsilon)}$) is as claimed in Theorem 6.1.

First, we focus on the correctness of the testers. Suppose that the input function $f$ is Lipschitz. Since Lipschitz functions do not have any violated pairs, Algorithm 5 may only reject $f$ in Step 1. When $R = nd$ this happens with probability 0, since Lemma 6.5 guarantees that $r \leq \text{ImD}(f) \leq nd$; Algorithm 5 with $R = nd$ is then a 1-sided error tester. Now consider the case when $R = n\sqrt{d\ln(84/\epsilon)}$. By the second part of Lemma 6.5 and Lemma 6.4, with probability at least $2/3$ we have $r \leq R$ (notice that SAMPLE-DIAMETER is evoked with parameter $\epsilon/2$). Thus, Algorithm 5 with $R = n\sqrt{d\ln(84/\epsilon)}$ accepts the Lipschitz function $f$ with probability at least $2/3$.

Now consider the case when $f$ is $\epsilon$-far from Lipschitz. We show that with probability at least $2/3$, $f$ is rejected in some iteration of Step 2; this part of the analysis is independent of the setting of $R$. Let $E$ be the event that $r$ is a good estimate, namely that $r \geq \text{ImD}_{r/2}(f)$; from part 1 of Lemma 6.5, $E$ holds with probability at least $5/6$. Then $f$ is rejected with probability at least $\Pr(\text{Step 2 rejects and } E \text{ holds}) \geq \Pr(\text{Step 2 rejects } | E)(5/6)$. Conditioned on $E$ (or more precisely conditioning on a realization of $r$ satisfying $E$), Lemma 6.7 gives that the probability $p$ of rejection on a single execution of Step 2 rejects is at least $\epsilon/40dr\log\min\{r,n\}$. Therefore, we using the standard approximation $(1 - x) \leq e^{-x}$ valid for all $x$, we obtain that $\Pr(\text{Step 2 rejects } | E)$ is at least $1 - (1 - p)^\ell \geq 4/5$; it then follows that the probability of rejection by the procedure is at least $(5/6)(4/5) = 2/3$.

Finally, we analyze the running time of the testers. Observe that since all operations performed by the algorithms (computing the maximum and simple comparisons) take time at most linear in the number of queries, the time complexity is the same as query complexity (in the model where each required random number is generated in one step). It remains to analyze the query complexity of Algorithm 5 for both settings of $R$. First, SAMPLE-DIAMETER in Step 1 makes $O(1/\epsilon)$ queries. Each iteration of Step 2 makes only 2 queries. Finally, by construction of the estimator, $r \leq \text{ImD}(f)$ and whenever the for loop is executed we also have $r \leq R$. This gives that the total number of iteration of the for loop is executed is at most $(80d \min \{\text{ImD}(f), R\} \log \min \{\text{ImD}(f), R, n\})/\epsilon$, so the total number of queries made by Algorithm 5 is upper bounded by

$$O\left(\frac{d \min \{\text{ImD}(f), R\} \log \min \{\text{ImD}(f), R, n\}}{\epsilon}\right).$$
Our choices of $R$ give the desired query complexity, concluding the proof of Theorem 6.1.

6.5 Testers for functions with range $\delta \mathbb{Z}$

In this section, we discuss modifications in the proofs required to obtain the desired testers for functions $f : [n]^d \rightarrow \delta \mathbb{Z}$ with $\delta \in (0, 1]$.

6.5.1 Modifications to Section 6.2

First, the main product of Section 6.2, namely, Theorem 6.3, holds as stated for the more general functions $f : [n] \rightarrow \delta \mathbb{Z}$ with $\delta \leq 1$. To prove it, we start by changing the definition of the basic operator to modify the values of the function by $\pm \delta$.

**Definition 6.8** (Basic operator). Given $f : [n]^d \rightarrow \delta \mathbb{Z}$ and $x, y \in [n]^d$ where $|x - y|_1 = 1$ and vertex names $x$ and $y$ are chosen so that $f(x) \leq f(y)$, the basic operator $B_{x,y}$ works as follows: If the pair $(x, y)$ is not violated by $f$ then $B_{x,y}[f]$ is identical to $f$. Otherwise, $B_{x,y}[f](x) = f(x) + \delta$ and $B_{x,y}[f](y) = f(y) - \delta$.

Now on to procedures **LinePass** and **BubbleSmooth**. **LinePass** now uses the new basic operator defined above, but no other changes are required in these procedures. However, notice that whenever **LinePass** $(i)$ is applied to a function $f$, now there is transfer of $\delta$ units between $i$ and a node $j$ (recall that when $\delta = 1$, one unit is transferred). Moreover, this node $j$ that participates in this transfer is the largest index such that the sequence $f(i), f(i + 1), \ldots, f(j)$ is: (i) monotone and (ii) consecutive terms differ by exactly 1; again this definition coincides with the one presented in Section 6.2 when $\delta = 1$.

The definition of transfer graph is unchanged, but its key property (Lemma 6.1) is scaled by a factor of $\delta$.

**Lemma 6.8** (Key property of transfer graph). Let $f : [n] \rightarrow \delta \mathbb{Z}$ be an input function given to **BubbleSmooth**. Then for each edge $e_s = (x, y)$ of the final transfer graph $T^*$, the following holds: $\text{vs}_f(x, y) \geq \delta(\text{deg}^*_s(x) + \text{deg}^*_s(y) - 1)$.

Naturally, Claim 6.1 used to prove Lemma 6.1 is also scaled accordingly; again the proof of Lemma 6.8 follows directly from the claim below.
Claim 6.3 (Invariant for phase \( i \)). Let \( f : [n] \to \delta \mathbb{Z} \) be an input function given to BubbleSmooth. At every point during the execution of BubbleSmooth(\( f \)), for each edge \( e_s = (x, y) \) of the transfer graph \( T \),

\[
f(x) - f(y) \geq \delta (\text{deg}_s(x) + \text{deg}_s(y) - 1) + |x - y|.
\]

Moreover, for each phase \( i \in [n - 1] \), after each execution of LinePass(\( i \)), for each edge \( e_s \) incident on vertex \( i \), the following (stronger) condition holds:

if the edge \( e_s = (i, j) \), i.e., it is outgoing from \( i \), then \( f(i) - f(j) \geq \delta (\Delta_i + \text{deg}_s(j) - 1) + |i - j| \);

if the edge \( e_s = (j, i) \), i.e., it is incoming into \( i \), then \( f(j) - f(i) \geq \delta (\Delta_i + \text{deg}_s(j) - 1) + |i - j| \).

The proof of Claim 6.1 can be used almost directly to prove Claim 6.3, the only modifications required being the following. First, (6.1) now becomes

\[
f^-(i) - f^-(j) \geq \delta (\Delta_i - \text{deg}^-_i (i)) + |i - j|
\]

(because each transfer lowers the violation score of the pair \((i, i + 1)\) by at least \( \delta \)). Since each transfer moves \( \delta \) units of mass instead of 1, equations (6.2) and (6.3) become respectively \( f(i) = f^-(i) + \delta \text{deg}^-_i (i) \) and \( f(j) = f^-(j) - \delta \text{deg}^-_i (j) \). For the same reason, equation (6.4) becomes \( f(i) = f^-(i) + \delta \text{deg}^-_i (i) \) and \( f(j) = f^-(j) + \text{deg}_j^- (j) \).

The rest of the proof can be used exactly as in the case \( \delta = 1 \) to obtain Claim 6.3.

With Lemma 6.8 at hand, we can prove Theorem 6.3 for functions \( f : [n] \to \delta \mathbb{Z} \) just as before. Indeed, the proof of part 1 of the theorem requires no changes. For part 2, we note that if \( f' \) is the (Lipschitz) function after the application of BubbleSmooth to \( f \), then \( |f - f'|_1 = \delta |E| \), where \( E \) is the set of edge in the final transfer graph. Moreover, using the greedy procedure as before, we can obtain a matching in the final transfer graph with violation score at least \( \delta |E| \); part 2 of Theorem 6.3 then follows from Lemma 6.2.

6.5.2 Modifications to Section 6.3

Technically, the proof of dimension reduction makes use of Claim 6.2. We present the corresponding lemma for the modified definition of the basic operator below and present
its proof. This lemma is sufficient to prove Item 3 of Lemma 6.3 for the modified definition of basic operator; the proofs of Items 1 and 2 hold as before.

6.5.2.1 Modifications to the basic operator on a square

Claim 6.4 (Properties of modified basic operator on a square). Consider a function \( f : \{x_t, x_b, y_t, y_b\} \rightarrow \delta \mathbb{Z} \) where vertices are labels of the four vertices of the square \( \{0, 1\}^2 \). (See Figure 6.2.) Let \( f' \) be the function obtained by applying modified basic operators \( \mathbb{B}_{x_t,y_t} \) and \( \mathbb{B}_{x_b,y_b} \) along the horizontal edges. Then the following holds for the vertical edges.

1. \( vs_f(x_t, x_b) + vs_f(y_t, y_b) \leq vs_f(x_t, x_b) + vs_f(y_t, y_b). \) (See proof of Lemma 3.3 of Chapter 3.)

2. \(|f'(x_t) - f'(x_b)| + |f'(y_t) - f'(y_b)| \leq |f(x_t) - f(x_b)| + |f(y_t) - f(y_b)|.\)

Proof. If neither horizontal edge is violated then \( f' = f \), and we are done. So assume w.l.o.g. \( \{x_t, y_t\} \) is violated such that \( f(y_t) \geq f(x_t) + 1 + \delta \). Then \( \mathbb{B}_{x_t,y_t} \) increases \( f(x_t) \) by \( \delta \) and decreases \( f(y_t) \) by \( \delta \) leading to Inequality (i): \( f'(y_t) \geq f(x_t) + 1 \).

If the absolute difference on the value of both vertical edges do not increase, we are also done. So assume w.l.o.g. that the absolute difference of the values of the left vertical edge increases strictly, namely \( |f'(x_b) - f'(x_t)| = |f(x_b) - f(x_t)| + \Delta \) for \( \Delta \in \{\delta, 2\delta\} \). This implies that \( \mathbb{B}_{x_b,y_b} \) did not increase the value at \( x_b \) and also that the following holds: (ii) \( f(x_b) \leq f(x_t) \). The former also implies that: (iii) \( f'(y_b) \leq f'(x_b) + 1 \). The definition of \( \Delta \) further gives that: (iv) \( f'(x_b) = f(x_b) - (\Delta - \delta) \) and (v) \( f'(y_b) = f(y_b) + (\Delta - \delta) \).

Using inequalities (i), (ii), (iv) and (iii) we get

\[
 f'(y_t) \geq f(x_t) + 1 \geq f(x_b) + 1 = f'(x_b) + (\Delta - \delta) + 1 \geq f'(y_b) + (\Delta - \delta),
\]

hence \( f'(y_t) \geq f'(y_b) \). Further using equation (v) we get

\[
|f'(y_t) - f'(y_b)| = f'(y_t) - f'(y_b) = f(y_t) - \delta - (f(y_b) + (\Delta - \delta)) \leq |f(y_t) - f(y_b)| - \Delta.
\]

Thus, the absolute difference of the values for the edge \( \{x_b, x_t\} \) increases by \( \Delta \) while the same for edge \( \{y_b, y_t\} \) decreases by at least \( \Delta \), proving the claim. \[\square\]
6.5.2.2 Remaining modifications

The proof of Theorem 6.4 for functions \( f : [n] \rightarrow \delta \mathbb{Z} \) follows exactly as before. Now Theorem 6.2 looses a factor of \( 1/\delta \) in the bound.

**Theorem 6.6** (Dimension reduction). *For all functions \( f : [n]^d \rightarrow \delta \mathbb{Z} \), the following holds:*

\[
\mathbb{E}_{g \leftarrow L_f} \left[ e^{Lip}(g) \right] \geq \frac{e^{Lip}(f)}{2 \cdot d \cdot \delta \text{ImD}(f)}.
\]

This follows directly from Theorem 6.4 and Observation 6.1.

6.5.3 Modifications to Section 6.4

No modifications are required in the procedure for estimating the image diameter of a function or for \textsc{Line-Tester}. However, the guarantee for the later given by Lemma 6.7 worsens by a factor of \( \delta \).

**Lemma 6.9.** Consider a function \( f : [n]^d \rightarrow \delta \mathbb{Z} \) that is \( \epsilon \)-far from Lipschitz and consider a real number \( r \geq \text{ImD}_{\epsilon/2}(f) \). For a random line \( g \leftarrow L_f \), the probability that \textsc{Line-Tester}(\( g, r \)) rejects is at least \( \frac{e^\delta}{40dr \log \min\{r,n\}} \).

The proof of this lemma is exactly as before, only now we construct the truncation \( f' \) of the original function \( f \) by taking \( a < b \) in \( \delta \mathbb{Z} \) (as opposed to in \( \mathbb{Z} \)) such that \( |a - b| \leq \text{ImD}_{\epsilon/2}(f) \) and at most \( \epsilon n^d/2 \) points \( x \in [n]^d \) have \( f(x) \notin [a, b] \); then define \( f'(x) = a \) if \( f(x) < a \), \( f'(x) = f(x) \) if \( f(x) \in [a, b] \) and \( f'(x) = b \) if \( f(x) > b \) as before, and the rest of the proof carries through (now using Theorem 6.6).

The only modification in our final tester, Algorithm 5, is in the number of iterations of the \textbf{for} loop, which is multiplying by \( 1/\delta \). The proof of the correctness and running time of this modified algorithm over function \( f : [n]^d \rightarrow \delta \mathbb{Z} \) follow exactly as before. This concludes the proof of Theorem 6.1 for arbitrary \( \delta \in (0,1) \).
Chapter 7

A deterministic local filter for the Lipschitz property

In this chapter, we prove Theorem 7.1, giving local filter of the Lipschitz property for function $f : \mathcal{H}_{n,d} \rightarrow \mathbb{R}$. Our filter is deterministic and nonadaptive.

**Theorem 7.1** (Local Lipschitz filters for Hypergrid). There is a deterministic nonadaptive local Lipschitz filter for functions $f : [n]^d \rightarrow \mathbb{R}$ with running time (and the number of lookups) $O((\log n + 1)^d)$ per query.

Rest of the chapter is organized as follows. We start by defining lookup graphs in Definition 7.2. In Lemma 7.1, we show how to use them to construct Lipschitz filters. Finally, we construct lookup graphs for the line and the hypergrid in Lemma 7.2. Lemmas 7.1 and 7.2 imply Theorem 7.1.

**Definition 7.1.** 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 7.2** (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 7.1** (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 $\text{reach-deg}(H)$ and running time $O(\text{reach-deg}(H) \cdot \text{out-deg}(H))$.

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 2.6.)

**Algorithm 6:** \textsc{Filter}_H(f, x)

1. If $\mathcal{N}(x)$ is empty, output $g(x) = f(x)$.
2. For each vertex $z$ in $\mathcal{N}(x)$, recursively compute $g(z) = \text{Filter}_H(f, z)$.
3. If setting $g(x) = f(x)$ makes $g$ Lipschitz on $\mathcal{N}^*(x)$, output $g(x) = f(x)$.
4. Otherwise, output $g(x) = \max_{z \in \mathcal{N}(x)} (g(z) - d_G(x, z))$.

We proceed to prove correctness of the filter. The recursion on Line 2 terminates because $H$ is acyclic. Function $g$ returned by the filter is identical to the input function $f$, provided that $f$ is Lipschitz.

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

**Claim 7.1.** 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 $\{x, y\}$. \qed

To show that function $g$ returned by Algorithm 6 is Lipschitz, it is sufficient, by Claim 7.1, 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)| <
The induction hypothesis gives us that \( g \) is Lipschitz on \( \mathcal{R}^*(z) \) for all \( z \in \mathcal{R}(x) \). Then Claim 7.1 implies that \( g \) is Lipschitz on \( \mathcal{R}(x) \). Lines 3 and 4 in Algorithm 6 ensure that \( g \) is Lipschitz on \( N^*(x) \). 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 \( \text{reach-deg}(H) \). 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 \( O(\text{reach-deg}(H) \cdot \text{out-deg}(H)) \). This is because the number of recursive calls to the filter is at most \( \text{reach-deg}(H) \) and the time spent on each call is \( O(\text{out-deg}(H)) \).

**Lemma 7.2 (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 is 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 7.2. 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, \( x \leq \text{LCA}(x, y) \leq y \). 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 $H_{n,d}$, we use the fact that $H_{n,d}$ is the Cartesian product of $d$ line graphs $L_n$. Claim 7.2 shows that the strong product (Definition 7.4) of lookup graphs is a lookup graph of the Cartesian product graph. We first define the Cartesian and strong graph products.

**Definition 7.3** (Cartesian graph product). Given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the Cartesian graph product, denoted by $G_1 \times G_2$, is a graph with the vertex set $V_1 \times V_2$. It contains an edge from $(x_1, x_2)$ to $(y_1, y_2)$ if and only if $x_1 = y_1$ and $(x_2, y_2) \in E_2$, or $(x_1, y_1) \in E_1$ and $x_2 = y_2$.

**Definition 7.4** (Strong product). Given directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the strong product of $G_1$ and $G_2$, denoted $G_1 \square G_2$, is a graph with the vertex set $V_1 \times V_2$ and the edge set

$$\{(x_1, x_2), (y_1, y_2)\} \mid (x_1, x_2) \in V_1 \times V_2, (y_1, y_2) \in V_1 \times V_2, (x_1, x_2) \neq (y_1, y_2) \}.$$

We use the following fact about shortest paths in Cartesian graph products to prove Claim 7.2.

**Fact 7.1.** 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 7.2.** 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, $\text{out-deg}(H) \leq (\text{out-deg}(H_1) + 1)(\text{out-deg}(H_2) + 1)$ and $\text{reach-deg}(H) \leq (\text{reach-deg}(H_1) + 1)(\text{reach-deg}(H_2) + 1)$.

**Proof.** The strong product of two DAGs is a DAG. Next we prove that $H$ satisfies the two properties of Definition 7.2. Given vertices $(x_1, x_2), (y_1, y_2)$ of $G$, let $z_1 \in \mathcal{R}^*_H(x_1) \cap \mathcal{R}^*_H(y_1)$ 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 7.1,
\((z_1, z_2)\) 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 \(\mathcal{R}_H(x) = \mathcal{R}^*_H(x_1) \times \mathcal{R}^*_H(x_2) \setminus \{x\}\). If \(y_1 \in \mathcal{R}_H(x_1)\), 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 7.1, 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 \(\text{out-deg}(H)\) and \(\text{reach-deg}(H)\) 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 \(L_n\) constructed earlier. The lookup graph \(H\) for \(H_{n,d}\) 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 7.2 establish the desired properties of \(H\). This completes the proof of Lemma 7.2.

Theorem 7.1 follows from Lemmas 7.1 and 7.2.
Chapter 8

Limitations of local filters of Lipschitz and monotone functions

8.1 Overview

In this chapter, we consider local $a$-filters, which relaxes the definition of local filters given in Definition 2.5. The relaxation allows answers to the queries to the filter to deviate from the true value by a small additive amount, even if the input function $f$ satisfies the property. Specifically, $a$-filters do not require that the reconstructed function $g$ be identical to the input function $f$ (with high probability) if the input $f$ satisfies the property under consideration. Instead, it is enough to have $f$ and $g$ be close in the following sense: with high probability, for all $x$ in the domain, $|f(x) - g(x)| \leq a$. Our main results, described in Section 8.1.2 and more formally in Section 8.2, are that even such relaxed filters need to perform a number of lookups exponential in the dimension $d$ in order to reconstruct a Lipschitz (respectively, monotone) function on domain $\{0, 1\}^d$.

Before describing our results and techniques in detail, we discuss some previous work on this problem.

8.1.1 Related work

Despite the fact that local filters have been thoroughly studied, lower bounds for general (not necessarily distance-respecting) adaptive filters remained a challenge, prior to our work. Saks and Seshadhri [SS10] present a distance-respecting local filter for mono-
tonicity of functions $f : [n]^d \rightarrow \mathbb{R}$ with running time $(\log n + 1)^{O(d)}$ per query. For monotonicity of functions $f : \{0, 1\}^d \rightarrow \mathbb{R}$, no nontrivial (i.e., performing $o(2^d)$ lookups per query) filter is known. Saks and Seshadhri also show that a distance-respecting local filter for monotonicity on the domain $\{0, 1\}^d$ must perform $2^{\Omega(d)}$ lookups per query. This lower bound crucially uses the fact that the filter is distance respecting, and does not apply to general local filters (even when no additive error is allowed). Bhattacharyya et al. [BGJ+12a] studied lower bounds for local monotonicity filters which are not necessarily distance-respecting. However, their super-polynomial lower bounds only hold for nonadaptive filter. For the domain $\{0, 1\}^d$, Bhattacharyya et al. show that nonadaptive filters must perform $\Omega(2^{\alpha d})$ lookups per query in the worst case, where $\alpha \approx 0.1620$. For adaptive filters, their bound quickly degrades with the number of lookups performed to incomparable points in the domain ($x, y \in [n]^d$ are comparable if $x \preceq y$ or $y \preceq x$ and incomparable otherwise). Specifically, their lower bounds for adaptive filters is $\Omega\left(\frac{2^{\alpha d}}{d}\right)$, where $\ell$ is the number of lookups to points incomparable to $x$ made on query $x$; for arbitrary adaptive filters, this degrades to $\Omega(d)$. Prior to our work, no super-polynomial lower bound for adaptive local monotonicity filter was known.

For the Lipschitz property, in Chapter 7, we presented a deterministic nonadaptive local filter that runs in time $O((\log n + 1)^d)$ per query. In the work (see [JR11]) in which we established this upper bound, we also show that the lower bound from [BGJ+12a] for nonadaptive filters, with the same statement, applies to nonadaptive local filters of the Lipschitz property. (We do not include this lower bound in this thesis because the current chapter supersedes this result.)

Previous work left open whether it is possible to obtain (adaptive and not necessarily distance-respecting) local filters for monotonicity and Lipschitz properties that make only $\text{poly}(n,d)$ lookups per query.

### 8.1.2 Our results

Our first result gives a lower bound on the number of lookups required by a Lipschitz $\alpha$-filter.

**Theorem 8.1** (Limitations of Lipschitz filters). Consider the Lipschitz property of functions $f : \{0, 1\}^d \rightarrow \mathbb{R}$ and any (randomized) local (not necessarily distance-respecting) $\frac{d}{402}$-filter for this property. Then there is a function $f$ and a query $x$ where, with constant
probability, this filter makes $2^{\Omega(d)}$ lookups.

The additive error $a = d/402$ in the theorem above is as large as possible up to a constant factor: the trivial filter that outputs $F(x) = (f(0) + f(1))/2$, where $0$ and $1$ are all-0 and all-1 vectors, respectively, is a local $\frac{d}{2}$-filter.\textsuperscript{1} To see this, note that (i) the reconstructed function $F(x)$ is Lipschitz and (ii) if the input function $f(x)$ is Lipschitz then $|F(x) - f(x)| = \frac{1}{2}|f(0) + f(1) - 2f(x)| \leq \frac{1}{2}(|f(0) - f(x)| + |f(1) - f(x)|) \leq \frac{1}{2}(|0 - x|_1 + |1 - x|_1) = \frac{d}{2}$ for every $x \in \{0,1\}^d$.

For monotonicity, we prove an analogous theorem with no upper bound on $a$. This is explained by the fact that monotonicity is determined by the order of the values at different points and not their magnitudes. To calibrate the additive error, we state the next theorem for functions with bounded range, namely, $[0, 2a + 1]$. The additive error in the theorem is also tight because for functions with that range, the trivial filter above that outputs $F(x) = (f(0) + f(1))/2$ is a local $(a + \frac{1}{2})$-filter.

**Theorem 8.2 (Limitations of monotonicity filters).** Consider the monotonicity property of functions $f : \{0,1\}^d \to [0,2a+1]$ and any (randomized) local $a$-filter for this property. Then there is a function $f$ and query $x$ where, with constant probability, this filter makes $2^{\Omega(d)}$ lookups.

To introduce the ideas used in the proofs, we focus for now on deterministic filters. To obtain lower bounds for nonadaptive filters in [BGJ+12a, JR11], the authors construct two collections of ‘hard functions’ $f(x,y)$ and $f(x,y)$ (satisfying the Lipschitz property) indexed by $x,y \in \{0,1\}^d$. They show that if a local filter works correctly on $f(x,y)$ and $f(x,y)$, as well as on a suitably defined function $h(x,y)$ (violating the Lipschitz property on $(x,y)$), the lookups made on queries $x$ and $y$ need to have a structured interaction. (Note that in this case the lookups are independent of the input function because the filter is nonadaptive.) More precisely, they construct a graph over $\{0,1\}^d$ based on these interactions and show that it is a 2-transitive-closure-spanner (2-TC-spanner) [BGJ+12b] for the hypercube. (2-TC-spanner is defined in Definition 2.9.) Using the lower bound on the size of a 2-TC-spanner for the hypercube from [BGJ+12a], it can be shown that any non-adaptive filter must use exponential lookups on one of the query points.

In the case of adaptive filters one cannot assume that the lookups made on a given query point are independent of the input function. One simple idea to try to overcome

\textsuperscript{1}In order to simplify the presentation, we did not attempt to optimize this constant factor.
this obstacle is to consider, for each query \( x \), the union of the lookups made on query \( x \) over all possible choice of hard functions. One can then try to apply the lower bound approach discussed in the previous paragraph. In fact this union of lookups still has strong interactions that imply a 2-TC-spanner. The problem is that this clearly overcounting the number of lookups made by the filter on a single given function on query \( x \). Due to the large number of ‘hard functions’ considered in \([BGJ^{+}12a, JR11]\), this overcounting makes the bound coming from the 2-TC-spanners vacuous for adaptive filters; this is where the factor \( 2^\ell \) lost in \([BGJ^{+}12a]\) mentioned above comes from.

In order to remedy this, we build a collection of hard functions which are much ‘smoother’ than those from \([BGJ^{+}12a, JR11]\). This allows us to use fewer functions. However, it comes at a cost: the interactions of the lookups caused by these functions are not as structured as before and do not imply a 2-TC-spanner. We introduce a type of directed graph called \( c \)-connector (Definition 8.3) which captures lookup interactions. As discussed in Section 8.3, our transformation to \( c \)-connectors preserves information on whether \( x \) is looked up on query \( y \) or vice versa, while this information is lost in the transformation to 2-TC-spanners in \([BGJ^{+}12a, JR11]\). A \( c \)-connector would be a relaxation of 2-TC-spanners if not for the direction of the edges. A \( 1 \)-connector for a set \( P \) of pairs \( x \preceq y \) is a directed graph with the vertex set \( \{0, 1\}^d \) satisfying the following requirement: for every pair \( (x, y) \in P \), there is a common out-neighbor \( z \) of \( x \) and \( y \), such that \( z \preceq y \) and \( \|z\|_1 \geq \|x\|_1 \). More generally, for a \( c \)-connector, the requirements on the common out-neighbor \( z \) are that fewer than \( c \) coordinates are 1 in \( z \) and 0 in \( y \) \( |z \setminus y| < c \) and that \( \|z\|_1 > \|x\|_1 - c \). Notice that the requirements on \( z \) only depend on the weight of \( x \), not \( x \) itself, potentially allowing more points \( x \) connect to nodes \( y \) via the same out-neighbor \( z \) than can be done in a 2-TC-spanner. Nevertheless, we can argue that a \( c \)-connector has a large maximum outdegree, which relates to the lookup complexity. Indeed, one of the key ingredients for our lower bound is recognizing the limitations of 2-TC-spanners in this context and finding a combinatorial structure with the right amount of flexibility. Given the importance of TC-spanners (see \([Ras10]\) for a survey), \( c \)-connectors might find use outside of this work.

**Organization.** Section 8.2 gives basic definitions and states our main results formally. In Section 8.3, we define \( c \)-connectors, the graph objects on which our lower bounds are based. In Sections 8.4 and 8.5, we develop a connection between \( c \)-connectors and
local filters for the Lipschitz property and monotonicity. In Section 8.6, we bound the outdegree of \( c \)-connectors. The final proof of the theorems above appears in Section 8.7 and consists of putting these two parts together.

### 8.2 Definitions and formal statement of results

Given a point \( x \in \{0, 1\}^d \), we use \( x_i \) to denote its \( i \)th coordinate and \( |x| \) to denote its Hamming weight, that is, \( |x| = \sum_i x_i \). We identify each point \( x \in \{0, 1\}^d \) with the subset of coordinates where it takes value 1, namely, \( \{i : x_i = 1\} \). This gives meaning to expressions like \( x \subseteq y \), \( x \cap y \), \( x \cup y \) and \( x \setminus y \) for \( x, y \in \{0, 1\}^d \). For \( x \in \{0, 1\}^d \), the Hamming weight \( |x| \) coincides with the cardinality of the set associated with \( x \).

We now provide a formal definition of local \( a \)-filters that allow additive error \( a \). It is stated for a general property \( P \) of functions with domain \( D \); in our case, \( P \) will be either the Lipschitz property or monotonicity.

**Definition 8.1 (Local \( a \)-filter).** Let \( P \) be a property of functions \( f : D \to R \) for some \( R \subseteq \mathbb{R} \). A local \( a \)-filter for \( P \) with error probability \( \delta \) is a randomized algorithm which is given black-box access to a function \( f : D \to R \) together with a query point \( x \in D \). For each random seed \( \sigma \) in the algorithm’s probability space \((\Omega, \Pr)\), the filter obtains the value of \( f \) on a sequence of points \( L(\sigma, f, x) = \{y_1, y_2, \ldots, y_k\} \), called lookups, (where the choice of \( y_i \) depends only on \( x, \sigma \) and \( f(y_1), f(y_2), \ldots, f(y_{i-1}) \)) and outputs a reconstructed value \( F(\sigma, f, x) \) for \( x \) solely based on the values of \( f \) at \( L(\sigma, f, x) \). The reconstructed function \( F_{\sigma,f} : D \to R \) given by \( F_{\sigma,f}(x) = F(\sigma, f, x) \) must obey two conditions: (i) \( F_{\sigma,f} \) satisfies property \( P \) for all functions \( f \) and all random seeds \( \sigma \); (ii) if \( f \) satisfies property \( P \) then for all \( x \in D \) we have \( \Pr_\sigma(F_{\sigma,f}(x) \in [f(x) - a, f(x) + a]) \geq 1 - \delta \).

Notice that requirement (ii) in this definition is weaker than requiring that “if \( f \) satisfies property \( P \) then \( \Pr(\forall x \in D, F_{\sigma,f}(x) \in [f(x) - a, f(x) + a]) \geq 1 - \delta \)”; therefore, we manage to obtain lower bounds for a more general class of filters. As a notational remark, we usually omit the probability space and denote a local \( a \)-filter by \((L, F)\).

The next observation captures the structural rigidity of local filters exploited in our lower bounds. It states that if functions \( f \) and \( g \) are identical on the lookups performed
on query $x$ when the input function is $f$, then the filter will perform the same lookups on $x$ for both $f$ and $g$ and, consequently, reconstruct the same value.

**Observation 8.1.** Let $(L, F)$ be a local $a$-filter. Then the following holds for every random seed $\sigma$ and query point $x$: if $f$ and $g$ are functions such that $f|_{L(\sigma,f,x)} = g|_{L(\sigma,f,x)}$, then $F(\sigma,f,x) = F(\sigma,g,x)$.

Now we restate Theorems 8.1 and 8.2, giving more details about parameters we obtain.

**Theorem 8.3.** Fix a non-negative constant $\delta$, consider a sufficiently large integer $d$ (depending on $\delta$) and let $a \in [0, d/402]$. Let $(L, F)$ be a local $a$-filter for the Lipschitz property with error probability $\delta$. Then there exists a function $f : \{0, 1\}^d \to \mathbb{R}$ and a query $x \in \{0, 1\}^d$ such that $\Pr_{\sigma}(|L(\sigma,f,x)| \geq 2^{0.009d}) \geq 1/2 - 1.1\delta$.

**Theorem 8.4.** Fix a non-negative constant $\delta$, consider a sufficiently large integer $d$ (depending on $\delta$) and let $a \geq 0$. Let $(L, F)$ be a local $a$-filter for monotonicity with error probability $\delta$. Then there exists a function $f : \{0, 1\}^d \to [0, 2a + 1]$ and a query $x \in \{0, 1\}^d$ such that $\Pr_{\sigma}(|L(\sigma,f,x)| \geq 2^{0.009d}) \geq 1/2 - 1.1\delta$.

### 8.3 $c$-Connectors

In this section, we formally introduce the notion of $c$-connectors. This combinatorial structure can be represented as a directed graph on the vertex set $\{0, 1\}^d$, where pairs of nodes need to share an out-neighbor with some prescribed properties. As we shall see next, $c$-connectors are related to 2-TC-spanners, although the full motivation for the exact definition will only become clear in Sections 8.4 and 8.5.

**Definition 8.2.** Let $X$ denote the set of points in $\{0, 1\}^d$ with Hamming weight exactly $d/3$ and let $Y$ denote the set of points in $\{0, 1\}^d$ with Hamming weight exactly $2d/3$. Also let $\mathcal{P}$ denote the set of comparable pairs $(x,y) \in X \times Y$, namely, such that $x \prec y$.

**Definition 8.3 ($c$-connector).** Fix $c \in \mathbb{N}$. Given a subset $\mathcal{P}'$ of $\mathcal{P}$, a digraph $G$ with the node set $\{0, 1\}^d$ is a $c$-connector for $\mathcal{P}'$ if for every $(x,y) \in \mathcal{P}'$ there exists $z \in \{0, 1\}^d$ with the following properties:

- (Connectivity) The arcs $(x, z)$ and $(y, z)$ belong to $G$. 

• \((Structure)\ |z \setminus y| < c\ and\ |z| > \frac{d}{3} - c.\)

A 2-TC-spanner (see also Definition 2.9) of the boolean hypercube (with the usual partial order) is a directed graph \(H\) on the node set \(\{0, 1\}^d\) with the property that for all \(x \prec y\) there is a point \(z\) satisfying \(x \preceq z \preceq y\), such that the arcs \((x, z)\) and \((z, y)\) belong to \(H\) [BGJ+12b]. If we reorient the arcs in a 2-TC-spanner of the hypercube, so that the nodes in \(Y\) only have outgoing arcs, we obtain a valid \(c\)-connector for every \(c \geq 1\): this is because the requirement \(x \preceq z \preceq y\) (in the definition of 2-TC-spanner) implies the structure requirement in a \(c\)-connector. Therefore, \(c\)-connectors relax 2-TC-spanners in two ways: first it requires that only pairs in \(P\) have a common neighbor with prescribed properties, and second it relaxes the required properties of this common neighbor. We remark that the direction of the arcs in \(c\)-connectors is important here, since in order to obtain the desired results we lower bound the outdegree. In contrast, in previous work [BGJ+12a, JR11] the information of whether point \(x\) was looked up on query \(y\) or vice versa was lost in the transformation to the corresponding 2-TC-spanner and the lower bound on the number of arcs, not the outdegree, was used. This is one of the changes that gives us stronger lower bounds.

**8.4 Lipschitz local filters imply \(c\)-connectors**

In this section we focus on the Lipschitz property. We construct a family of functions such that a local \(a\)-filter that works correctly on functions from the family must preform lookups corresponding to a \(c\)-connector. The idea is to start with a Lipschitz function \(f^0\) and then construct other Lipschitz functions \(f_y^c\) which agree with \(f^0\) on most points, but where \(f_y^c(y)\) is much larger than \(f^0(y)\). We argue that if a purported local \(a\)-filter makes only ‘local’ lookups when reconstructing at queries \(x\) and \(y\), then we can create a function that looks like \(f_y^c\) around \(y\) (so that the filter is fooled and returns \(F(y)\) in the range \(f_y^c(y) \pm a\)) and looks like \(f^0\) around \(x\) (so that the filter is fooled and returns \(F(x)\) in the range \(f^0(x) \pm a \ll f_y^c(y) \pm a\)). Thus, for the returned function, \(F(x)\) and \(F(y)\) are too far apart, ensuring that it is not Lipschitz.
8.4.1 Hard functions for filter

Recall from Definition 8.2 that $Y$ denotes the set of points in $\{0,1\}^d$ with Hamming weight exactly $d/3$. In order to construct these hard functions, for a point $y \in Y$ let $T_y = \{x \in \{0,1\}^d : x \subseteq y, |x| \geq d/3\}$. Define the function $f^0$ by $f^0(z) = \max\{|z|, d/3\}$ for all $z \in \{0,1\}^d$. Intuitively, for $c \in \mathbb{N}$ and $y \in Y$, we define the function $f^c_y$ as the smallest Lipschitz function which is at least $f^0 + c\chi_T$, where $\chi_T$ denotes the characteristic function of the set $T$. More specifically, we set $f^c_y(z) = \max\{|z| + c - |z \setminus y|, f^0(z)\}$ for all $z \in \{0,1\}^d$. A graphic representation of these functions is given in Figure 8.1. Clearly, function $f^0$ is Lipschitz. Next we prove that all functions $f^c_y$ are Lipschitz as well.

**Lemma 8.1.** For all $c \in \mathbb{N}$ and $y \in Y$ the function $f^c_y$ is Lipschitz.

**Proof.** Fix $c \in \mathbb{N}$ and $y \in Y$. Define $g(z) = |z| + c - |z \setminus y|$, so that $f^c_y = \max\{g, f^0\}$. Since $f^0$ is Lipschitz, and the maximum of two Lipschitz functions is a Lipschitz function, it is sufficient to show that $g$ is Lipschitz. Take $z, z' \in \{0,1\}^d$ such that $||z - z'||_1 = 1$; it remains to show that $|g(z) - g(z')| \leq 1$. Notice that either $z \subseteq z'$ or $z' \subseteq z$; without loss of generality assume the former. Since $|z| = |z'| - 1$ and $0 \leq |z' \setminus y| - |z \setminus y| \leq 1$, we obtain that $|g(z) - g(z')| \leq |z| - |z'| + |z' \setminus y| - |z \setminus y| \leq 1$. This concludes the proof of the lemma.

For a point $y \in Y$ and a constant $c \in \mathbb{N}$, let $T^c_y \subseteq \{0,1\}^d$ be the set of points $z$, such that $f^c_y(z) \neq f^0(z)$. Then $T^c_y = T_y$ and the set $T^c_y$ gets larger as $c$ increases: specifically, $T^c_y \subseteq T^{c'}_y$ for $c < c'$. The definitions of $f^c_y$ and $f^0$ directly give the following observation, which justifies the specific structure used in the definition of a $\alpha$-connector.

**Observation 8.2.** All elements $z$ in the set $T^c_y$ satisfy $|z \setminus y| < c$ and $|z| > \frac{d}{3} - c$.

8.4.2 Reconstruction of hard functions implies $\alpha$-connector

Now we show that if a local $\alpha$-filter is correct on the constructed functions, its lookups correspond to a $\alpha$-connector for the interesting pairs $P$ (recall that $P$ is the set of pairs $(x, y) \in X \times Y$ such that $x \prec y$). We start by essentially focusing on deterministic filters or, alternatively, by looking at a ‘good’ seed of a randomized filter. The analysis for randomized filters is based on the ability to pick a few of these good seeds and then analyzing the ‘union’ of the behavior of the filter running with these seeds.
Consider a local $\alpha$-filter $(L, F)$. Given points $x \in X$ and $y \in Y$, we say that a random seed $\sigma \in \Omega$ is good for $x$ and $y$ if $F_{\sigma, f^0}(x) \in [f^0(x) - a, f^0(x) + a]$ and $F_{\sigma, f^c}(y) \in [f^c_y(y) - a, f^c_y(y) + a]$. Given a seed $\sigma$ which is good for $x$ and $y$, we define the digraph $G^x_y = (\{0, 1\}^d, A^x_y)$ that captures the lookups made on queries $x$ and $y$. Specifically, the set $A^x_y$ consists of all the arcs $\{(x, z) : z \in L(\sigma, f^0, x) \cup \{x\}\}$ and $\{(y, z) : z \in L(\sigma, f^c_y, y) \cup \{y\}\}$.

**Figure 8.1.** Functions used in proof of Lemma 8.2.

![functions](image)

**Lemma 8.2** (Local filter implies $c$-connector). Consider a local $\alpha$-filter $(L, F)$ for the Lipschitz property and an integer $c > 2a$. For all $(x, y) \in \mathcal{P}$, if $\sigma \in \Omega$ is good for $x$ and $y$ then $G^x_y$ is a $c$-connector for $(x, y)$.

**Proof.** For the sake of contradiction suppose not. Unraveling the definitions and using Observation 8.2 this means that the sets $(L(\sigma, f^0, x) \cup \{x\}) \cap T^c_y$ and $(L(\sigma, f^c_y, y) \cup \{y\}) \cap T^c_y$ do not intersect. Then let $A, B$ be a partition of $T^c_y$ such that $A$ contains $(L(\sigma, f^0, x) \cup \{x\}) \cap T^c_y$ and $B$ contains $(L(\sigma, f^c_y, y) \cup \{y\}) \cap T^c_y$. Define the function $f$ such that $f|_A = f^0|_A$, $f|_B = f^c_y|_B$, and $f|_{\{0, 1\}^d \setminus (A \cup B)} = f^0|_{\{0, 1\}^d \setminus (A \cup B)} = f^c_y|_{\{0, 1\}^d \setminus (A \cup B)}$, where the last equation follows from the definition of $T^c_y$. (See Figure 8.1.) To reach a contradiction, we show that the filter does not reconstruct $f$ correctly.

Notice that $f^0|_{L(\sigma, f^0, x)} = f|_{L(\sigma, f^0, x)}$, so Observation 8.1 gives that $F(\sigma, f, x) = F(\sigma, f^0, x)$. Similarly, $f^c_y|_{L(\sigma, f^c_y, y)} = f|_{L(\sigma, f^c_y, y)}$ and hence $F(\sigma, f, y) = F(\sigma, f^c_y, y)$.

Now since $\sigma$ is good for $x$ and $y$, we have that $F(\sigma, f, x) = F(\sigma, f^0, x) \leq f^0(x) + a = \frac{d}{3} + a$ and $F(\sigma, f, y) = F(\sigma, f^c_y, y) \geq f^c_y(y) - a = \frac{2d}{3} + c - a$. Since $c > 2a$ we get
\[ F(\sigma, f, y) - F(\sigma, f, x) > d/3 = \|x - y\|_1, \]
and hence the function \( F_{\sigma,f} \) is not Lipschitz; this contradicts that \((L, F)\) is a local \( a \)-filter and concludes the proof. \qedhere

Consider two subsets \( P_1, P_2 \) of \( P \). Notice that if \( G_1 \) is a \( c \)-connector for \( P_1 \) and \( G_2 \) is a \( c \)-connector for \( P_2 \) then the graph formed by the union of (the arcs of) \( G_1 \) and \( G_2 \) is a \( c \)-connector for \( P_1 \cup P_2 \). We remark that when we take this union we do not add parallel arcs. This directly gives the following result.

**Corollary 8.1.** Consider a local \( a \)-filter \((L, F)\) for the Lipschitz property and an integer \( c > 2a \). Suppose that for each \((x, y) \in P\) there is a random seed \( \sigma(x, y) \in \Omega \) which is good for \( x \) and \( y \). Then the graph obtained as the union of the graphs \( \{G_{\sigma(x,y)}^{xy}(x,y)\}_{(x,y) \in P} \) is a \( c \)-connector for \( P \). Moreover, this graph has outdegree at most

\[
\max \left\{ \max_{x \in X} \left\{ |\bigcup_y L(\sigma(x, y), f^0, x)| \right\}, \max_{y \in Y} \left\{ |\bigcup_x L(\sigma(x, y), f^c_y, y)| \right\} \right\} + 1. \tag{8.1}
\]

Using this corollary, we show that a local \( a \)-filter with small ‘average’ number of lookups implies a \( c \)-connector for \( P \) with a small outdegree.

**Lemma 8.3.** Consider a local \( a \)-filter \((L, F)\) for the Lipschitz property with error probability \( \delta \) and an integer \( c > 2a \). Consider \( \alpha > 0 \) and let

\[
M = \max_{f, x} \Pr_{\sigma}(|L(\sigma, f, x)| > \alpha).
\]

If \( \delta + M < 1/2 \) then there is a \( c \)-connector for \( P \) with maximum outdegree at most

\[
2d\alpha / \log \left( \frac{1}{2\delta + 2M} \right) + 1.
\]

**Proof.** The idea is to construct, via the probabilistic method, a set \( S \subseteq \Omega \) of good seeds which attains a small value in (8.1). Given \((x, y) \in P\) define the event \( E_{x,y} \subseteq \Omega \) as the set of random seeds \( \sigma \) satisfying the following: (i) \( \sigma \) is good for \( x \) and \( y \); (ii) \( |L(\sigma, f^0, x)| \leq \alpha \) and \( |L(\sigma, f^c_y, y)| \leq \alpha \). Given the guarantee of the filter and the definition of \( M \), the complement of \( E_{x,y} \) holds with probability at most \( \gamma = 2\delta + 2M \).

Now let \( S \) be a random set obtained by picking independently and with replacement \( s \approxeq 2d / \log_2(1/\gamma) \) elements from \((\Omega, \Pr)\). For a given \((x, y) \in P\), it follows from the previous paragraph that the probability (over the construction of \( S \)) that \( S \) does not intersect \( E_{x,y} \) is at most \( \gamma^s \). Taking a union bound over all such pairs, the probability that
there is \((x, y) \in \mathcal{P}\) for which \(S\) does not intersect \(E_{x,y}\) is strictly less than \(2^{2d}\gamma^s = 1\). Therefore, there exists a realization \(\bar{S}\) of \(S\) which intersects all \(E_{x,y}\)’s.

Then for \((x, y) \in \mathcal{P}\) let \(\sigma(x, y)\) be a point in \(\bar{S} \cap E_{x,y}\). Since each \(\sigma(x, y)\) is good for \(x\) and \(y\) we can apply Corollary 8.1 using these seeds. By construction we have that for all \(x \in X\) the set \(\bigcup_y L(\sigma(x, y), f^0, x)\) has size at most \(|\bar{S}|\alpha\) and for all \(y \in Y\) the set \(\bigcup_x L(\sigma(x, y), f^c_y, y)\) has size at most \(|\bar{S}|\alpha\); this concludes the proof.

\[\square\]

### 8.5 Local filters for monotonicity imply 1-connectors

In this section, we consider the monotonicity property and show that again the lookups performed by a local \(a\)-filter give rise to a \(c\)-connector (in this case, with \(c = 1\)).

#### 8.5.1 Hard functions for filter

Again, we start by defining functions \(f^0\) and \(f^a\), such that if a local filter is correct on these functions, its lookups correspond to a 1-connector. Recall that for a point \(y \in Y\), we define \(T_y = \{x \in \{0, 1\}^d : x \subseteq y, |x| \geq d/3\}\). Define the function \(f^0\) by \(f^0(z) = 2a + 1\) if \(|z| \geq d/3\) and \(f^0(z) = 0\) if \(|z| < d/3\). For a point \(y \in Y\), we define the function \(f^a_y\) equal to \(f^0 - (2a + 1)\chi_{T_y}\), namely, \(f^a_y(z) = 2a + 1\) if \(z \geq d/3\) and \(z \notin T_y\) and \(f^a_y(z) = 0\) otherwise. A graphic representation of these functions is given in Figure 8.2. It can be easily verified that these functions are monotone.

**Lemma 8.4.** For all \(y \in Y\) and \(a \geq 0\), the functions \(f^0\) and \(f^a\) are monotone.

Notice that the functions \(f^0\) and \(f^a\) differ exactly on points in \(T_y\) and that \(T_y\) is the set of points which satisfy the structure property in the definition of a 1-connector.

#### 8.5.2 Reconstruction of hard functions implies 1-connector

Recall that \(\mathcal{P}\) is the set of comparable pairs \((x, y) \in X \times Y\) or, equivalently, pairs where \(x \in T_y\). Consider a local \(a\)-filter \((L, F)\) for monotone functions. As before, given \(x \in X\) and \(y \in Y\), we say that a random seed \(\sigma \in \Omega\) is *good* for \(x\) and \(y\) if \(F_{\sigma, f^0}(x) \in [f^0(x) - a, f^0(x) + a]\) and \(F_{\sigma, f^a_y}(y) \in [f^a_y(y) - a, f^a_y(y) + a]\). Given a seed \(\sigma\) which is good for \(x\) and \(y\), we define the digraph \(G^x_{\sigma} = (\{0, 1\}^d, A^x_{\sigma})\) in a way similar to what
Figure 8.2. Functions used in proof of Lemma 8.5 for $a = 0$. Observe $f^0_0(x) = 1$ and $f^1_1(y) = 0$.

we did in the previous section: we add to $A^x_y\sigma$ all the arcs $\{(x, z) : z \in L(\sigma, f^0_0, x) \cup \{x\}\}$ and $\{(y, z) : z \in L(\sigma, f^a_1, y) \cup \{y\}\}$.

Again the construction of our functions and the digraph $G^{x,y}_{\sigma}$ together with the behavior of local $a$-filters captured in Observation 8.1 give the following.

**Lemma 8.5.** Take $a \geq 0$ and consider a local $a$-filter $(L, F)$ for monotonicity. For any $(x, y) \in P$, if $\sigma \in \Omega$ is good for $x$ and $y$ then $G^{x,y}_{\sigma}$ is a $1$-connector for $(x, y)$.

**Proof.** For the sake of contradiction suppose not. Unraveling the definitions this means that the sets $(L(\sigma, f^0_0, x) \cup \{x\}) \cap T_y$ and $(L(\sigma, f^a_1, y) \cup \{y\}) \cap T_y$ do not intersect. Then let $A, B$ be a partition of $T_y$ such that $A$ contains $(L(\sigma, f^0_0, x) \cup \{x\}) \cap T_y$ and $B$ contains $(L(\sigma, f^a_1, y) \cup \{y\}) \cap T_y$. Define the function $f$ such that $f|_A = f^0|_A$, $f|_B = f^a|_B$, and $f|_{(0,1)^2 \setminus (A \cup B)} = f^0|_{(0,1)^2 \setminus (A \cup B)} = f^0|_{(0,1)^2 \setminus (A \cup B)}$ (the last equation follows from the fact that $f^0_0$ and $f^0$ only differ over $T_y$); to reach the desired contradiction, we show that the filter does not reconstruct $f$ correctly.

Notice that $f^0|_{L(\sigma, f^0_0, x)} = f|_{L(\sigma, f^0_0, x)}$, so Observation 8.1 gives that $F(\sigma, f, x) = F(\sigma, f^0_0, x)$. Similarly, $f^0|_{L(\sigma, f^a_1, y)} = f|_{L(\sigma, f^a_1, y)}$ and hence $F(\sigma, f, y) = F(\sigma, f^a_1, y)$.

Now since $\sigma$ is good for $x$ and $y$, we have that $F(\sigma, f, x) = F(\sigma, f^0_0, x) \geq f^0(x) - a = a + 1$ and $F(\sigma, f, y) = F(\sigma, f^a_1, y) \leq f^a_1(y) + a = a$. We then get that $F(\sigma, f, x) >
$F(\sigma, f, y)$ and hence the function $F_{\sigma,f}$ is not monotone. This contradicts that $(L, F)$ is a local $\alpha$-filter and concludes the proof of the lemma. \qed

Finally, we can utilize the same technique for finding a set of good seeds which achieve small value in (8.1) as done in Lemma 8.3 to obtain the desired connection between local $\alpha$-filters and 1-connectors for $P$.

**Lemma 8.6.** Take $\alpha \geq 0$ and consider a local $\alpha$-filter $(L, F)$ for monotone functions with error probability $\delta$. Consider $\alpha > 0$ and let

$$M = \max_{f,x} \Pr_{\sigma} (|L(\sigma, f, x)| > \alpha).$$

If $\delta + M < 1/2$ then there is a 1-connector for $P$ with maximum outdegree at most $2d\alpha / \log \left( \frac{1}{2\delta + 2M} \right) + 1$.

### 8.6 Lower bound on the maximum outdegree of a $c$-connector

Recall that $P$ is the set of pairs $(x, y) \in X \times Y$ such that $x$ and $y$ are comparable. We show a lower bound on the maximum outdegree of a $c$-connector for $P$. We remark that the constants in the bound are not optimized.

**Theorem 8.5.** Consider $d \geq 40, 200$ and let $c$ be an arbitrary integer in the range $[d/201, d/200]$. Then the maximum outdegree of any $c$-connector for $P$ is at least $2^{0.01d}$.

To prove this, let $G$ be a $c$-connector for $P$. Let $\tilde{T}_y^c = \{ z : |z \setminus y| < c, |z| > d/3 - c \}$ be the points which satisfy the structure property in Definition 8.3. Then $T_y \subseteq T_y^c \subseteq \tilde{T}_y^c$ for all $y \in Y$, and for $x \in T_y$ and $z \in \tilde{T}_y^c$ we have $x \cup z \in \tilde{T}_y^c$. We say that a pair $(x, y) \in P$ is covered by a point $z$ if $z \in \tilde{T}_y^c$ and the arcs $(x, z)$ and $(y, z)$ belong to $G$.

Each pair in $P$ needs to be covered by a point. For a fixed $x \in X$, the outdegree of $x$ in $G$ is at least the number of distinct points which are covering the pairs in $P$ containing $x$ (and similarly for a fixed $y \in Y$). The difficulty in lower bounding the outdegree of $x$ is that many pairs containing it can be covered by the same point. The heart of the argument is to show that no point can cover too many such pairs. It relies on the fact that the sets $\tilde{T}_y^c$ are ‘localized’. More precisely, consider a point $z$ and let $(x, y)$ be covered
by it. Notice that \( x \in T_y \) and \( z \in \tilde{T}_y \), hence \( x \cup z \in \tilde{T}_y \). If \( z \) is not near \( x \), namely, \( z \setminus x \) is large, then we argue that not too many points \( y \) satisfy \( x \cup z \in T_y \), given the localization of \( \tilde{T}_y \). On the other hand, if \( z \) is near \( x \) then there are not too many possibilities for \( x \) itself. Our bound is derived by putting these observations together.

In order to make the above argument work we divide the pairs in \( P \) into two groups based on the covers they have. Let \( \alpha \in [1/15, 1/14] \) be such that \( \alpha d \) is an integer, which exists since \( d \) is sufficiently large. For \((x, y) \in P \) and \( z \) that covers \((x, y)\), if \( |z \setminus x| \leq \alpha d \), then we say that \( z \) is near \( x \) and that \( z \) is a nearby cover of \((x, y)\). Let \( N \) denote the set of pairs \((x, y) \in P \) which have a nearby cover and let \( F = P \setminus N \) be the remaining pairs. For a fixed \( y \in Y \), define \( N_y \) as the pairs in \( N \) containing \( y \) and for \( x \in X \) define \( F_x \) as the pairs in \( F \) containing \( x \). Our goal is to upper bound \( N \) and \( F \). Towards this goal, define \( Z \subseteq \{0, 1\}^d \) to be the set of points which cover at least one pair in \( P \). Furthermore, for a given \( x \in X \), let \( Z_x \) denote the set of points which cover at least one pair in \( P \) containing \( x \). Define \( Z_y \) analogously. Observe that \( Z \) is the union of sets \( Z_x \) and \( Z_y \) over all \( x \in X \) and \( y \in Y \). The next two lemmas bound the sizes of \( N \) and \( F \), respectively. For each lemma, we give a proof sketch describing the main ideas of the proof. Since the actual proof is somewhat technical, we defer it to Section 8.6.1.

**Lemma 8.7.** Letting \( \Theta = d^2 \left( d^{3+\alpha d} \right) \left( 2d^{3+c} \right) \alpha d + c \), the number of pairs in \( N \) is at most \( |Y| \cdot \Theta \cdot \max_{y \in Y} \{ |Z_y| \} \).

**Proof sketch.** In order to upper bound \( N \) we start by arguing that, for a fixed \( y \in Y \), a point cannot be a nearby cover for many pairs \((x, y)\) in \( N_y \). To see this, take \( z \in Z_y \) and let \((x, y) \in P \) be such that \( z \) is a nearby cover for it. Then notice that \( x \) and \( z \) are very similar: \( |z \setminus x| \leq \alpha d \) and \( |x \setminus z| \leq \alpha d + c \); the first bound follows from the definition of a nearby cover and the second uses \( |z| \geq |x| - c \) from Observation 8.2. From these constraints, it follows that there are at most \( d^2 \left( d^{3+\alpha d} \right) \left( 2d^{3+c} \right) \alpha d + c \) possibilities for such \( x \)’s. Thus, for all \( y \in Y \) we have \( |N_y| \leq |Z_y| \cdot d^2 \left( d^{3+\alpha d} \right) \left( 2d^{3+c} \right) \alpha d + c \). Adding over all \( y \) gives the desired bound.

**Lemma 8.8.** Letting \( \Phi = d^2 \left( d^{3+c} \right) \left( 2d^{3+\alpha d} \right) \alpha d + c \), the number of pairs in \( F \) is at most \( |X| \cdot \Phi \cdot \max_{x \in X} \{ |Z_x| \} \).

**Proof sketch.** To upper bound the size of \( F \) we start by showing that, for a fixed \( x \in X \), a point cannot be a (non-nearby) cover for too many pairs in \( F_x \). To see this, take \( z \in Z_x \)
and suppose \((x, y) \in \mathcal{F}_x\) is covered by \(z\). Notice that \(x \cup z\) and \(y\) are very similar: \(|(x \cup z) \setminus y| \leq c\) and \(|y \setminus (x \cup z)| \leq d/3 - \alpha d + c\); the first bound follows from \(x \subseteq y\) and Observation 8.2, and the second further uses the fact that \(|x \cup z| \geq d/3 + \alpha d\) (since \(z\) is not a nearby cover). Then it is easy to see that there are at most \(d^2 \cdot (\frac{2d+\alpha d}{3}) \cdot (\frac{d-\alpha d+c}{3 \alpha d} + c)\) such \(y\)’s. Thus, for each \(x \in X\) we have \(|F_x| \leq |\mathcal{Z}_x| \cdot d^2 \cdot (\frac{2d+\alpha d}{3}) \cdot (\frac{d-\alpha d+c}{3 \alpha d} + c)\) and adding over all \(x\) gives the desired bound on \(\mathcal{F}\).

The maximum outdegree of the \(c\)-connector \(G\) is bounded from below by

\[
M \triangleq \max \{ \max_{x \in X} \{|\mathcal{Z}_x|\}, \max_{y \in Y} \{|\mathcal{Z}_y|\} \}.
\]

Since \(\mathcal{N}\) and \(\mathcal{F}\) partition the set of pairs \(\mathcal{P}\), we can add the bounds from Lemmas 8.7 and 8.8 and obtain

\[
M \geq \frac{|\mathcal{P}|}{(\frac{d}{d/3})(\Theta + \Phi)} = \frac{(\frac{2d/3}{d/3})}{\Theta + \Phi}.
\]

(8.2)

Standard computations (deferred to Section 8.6.2) can be used to lower bound the right-hand side of this expression by \(2^{0.01d}\). This concludes the proof of Theorem 8.5.

### 8.6.1 Estimates for Lemmas 8.7 and 8.8

Both estimates for Lemmas 8.7 and 8.8 come from the following technical claim.

**Claim 8.1.** Given \(t_1, t_2 \in [d]\) and a fixed \(u \in \{0, 1\}^d\) such that \(|u| \geq t_1\), define \(S(u, t_1, t_2, \eta)\) to be the set of vertices \(x \in \{0, 1\}^d\) such that \(|x| = t_2\) and \(|u \setminus x| \leq \eta\). Moreover, assume \(t_1\) (respectively, \(t_2\)) is at least \(2\eta + 2t_2 - d\) (respectively, \(\eta\)). Then, \(|S(u, t_1, t_2)| \leq d^2 \left(\frac{t_2+\eta}{\eta}\right) \left(\frac{d-t_1}{\eta+t_2-t_1}\right)\).

**Proof.** First, we show \(t_1 \leq |u| \leq \eta + t_2\). The lower bound is part of the statement of the claim. For the upper bound, using \(|u \setminus x| \leq \eta\) (respectively, \(|x| = t_2\)) from the statement of the claim, we get \(|u| = |u \setminus x| + |u \cap x| \leq \eta + |u \cap x| \leq \eta + |x| = \eta + t_2\). Next we show \(|x \setminus u| \leq \eta + t_2 - t_1\). Observe that \(|x \setminus u| = |x \cup u| - |u| = |x| + |u \setminus x| - |u|\). Using \(|u \setminus x| \leq \eta\), \(|x| = t_2\) and \(|u| \geq t_1\) from the statement of the claim, we get \(|x \setminus u| = |x| + |u \setminus x| - |u| \leq t_2 + \eta - t_1\), as required.
Therefore, for every \( x \in S(u, t_1, t_2, \eta) \), we have points \( r \) and \( a \) such that \( x = (u \setminus r) \cup a \) satisfying: (i) \( r \subseteq u \) and \( |r| \leq \eta \); (ii) \( a \cap u = \emptyset \) and \( |a| \leq \eta + t_2 - t_1 \). Since \( t_1 \leq |u| \leq \eta + t_2 \), there are

\[
\begin{align*}
&\text{at most } \sum_{i=0}^{\eta} \binom{t_2+\eta}{i} \leq d^{(t_2+\eta)/\eta} \text{ possibilities for } r \text{ and} \\
&\text{at most } \sum_{i=0}^{\eta+t_2-t_1} \binom{d-t_1}{i} \leq d^{(d-t_1)/(\eta+t_2-t_1)} \text{ possibilities for } a,
\end{align*}
\]

where in the above bounds we used the fact that \( t_1 \) (respectively, \( t_2 \)) is at least \( 2\eta + 2t_2 - d \) (respectively, \( \eta \)). Multiplying these terms gives the upper bound on \( |S(u, t_1, t_2, \eta)| \). \( \square \)

Estimate for Lemma 8.7. Recall that we have a fixed \( z \in \{0, 1\}^d \) and we want to upper bound the number of \( x \in X \) satisfying \( |z \setminus x| \leq \alpha d \) and \( |z| \geq |x| - c \) by \( \Theta = d^2 \left( \frac{d/3+\alpha d}{\alpha d} \right) \left( \frac{2d/3}{\alpha d+c} \right) \); this follows directly by applying Claim 8.1 with parameters \( u = z, t_1 = d/3 - c, t_2 = d/3 \) and \( \eta = \alpha d \).

Estimate for Lemma 8.8. Recall that now we have a fixed \( z \in \{0, 1\}^d \) and we want to upper bound the number of \( y \in Y \) satisfying \( |(x \cup z) \setminus y| \leq c \) and \( |x \cup z| \geq \frac{d}{3} + \alpha d \) by \( \Phi = d^2 \left( \frac{\frac{d}{3}+c}{c} \right) \left( \frac{\frac{d}{3}-\alpha d}{\frac{d}{3}-\alpha d+c} \right) \); this follows directly by applying Claim 8.1 with parameters \( u = x \cup z, t_1 = d/3 + \alpha d, t_2 = 2d/3 \) and \( \eta = c \).

8.6.2 Bounding \( \Theta + \Phi \)

In this section we show that

\[
\frac{\binom{2d/3}{d/3}}{d/3} \geq 2^{0.01d}.
\]

We start with three simple facts about the binomial coefficient \( \binom{a}{b} \) for integers \( a \geq b \): (i) \( \binom{a+1}{b+1} = \frac{a+1}{b+1} \binom{a}{b} \geq \binom{a}{b} \); (ii) \( \binom{a}{b} \leq \left( \frac{a}{b} \right)^b \); (iii) if \( b = a/2 \), then \( \binom{a}{b} \geq \frac{2^a}{a} \). We also observe that \( (1/x)^x \) is increasing in the range \( x \in (0, 1/4] \). Using this, it is already easy to see that by choosing \( \alpha \) and \( c/d \) small enough we can get \( \Theta \) and \( \Phi \) of the order \( \left( \frac{2d/3}{d/3} \right) O(2^{-\epsilon d}) \) for a small constant \( \epsilon > 0 \); we show that the choice of \( \alpha \) and \( c/d \) in the statement of the lemma works.

From observation (iii) above we have \( \binom{2d/3}{d/3} \geq \frac{2^{2d/3}}{d} \). Using observations (i) and (ii)
above and the bounds on \( \alpha, c \) and \( d \), we obtain the upper bound

\[
\frac{\Theta}{(\frac{2d}{3^d})} = \frac{d^2 (\frac{d}{\alpha} + (\frac{2d}{\alpha} + c)) (\frac{2d}{\alpha} + c)}{(\frac{2d}{3^d})} \leq d^3 2^{-\frac{2d}{3}} \left( \frac{d}{\alpha} + \frac{\alpha d}{\alpha d + c} \right) \left( \frac{2d}{3} + c \right)
\]

\[
\leq d^3 2^{-\frac{2d}{3}} \left( \frac{e(d/3 + \alpha d)}{\alpha d} \right)^{\alpha d} \left( \frac{2d}{\alpha d + c} \right) \left( \frac{2d}{3} + c \right)
\]

We have that

\[
\left( \frac{e(d/3 + \alpha d)}{\alpha d} \right)^{\alpha d} \leq \left( e(1 + \frac{1}{3\alpha}) \right)^{\alpha d} \leq 2^{0.2d}
\]

We also have that

\[
\left( \frac{2d/3 + c}{\alpha d + c} \right) \leq \left( \frac{2d/3}{\alpha d} \right) \left( \frac{2d/3 + c}{\alpha d} \right) \left( \frac{2d/3 + c}{\alpha d} \right)^{\alpha d} \left( \frac{2d/3 + c}{\alpha d} \right) \leq 2^{0.25d}
\]

Hence, we get

\[
\frac{\Theta}{(\frac{2d}{3^d})} \leq d^3 2^{-\frac{2d}{3}} 2^{0.45d} \leq \frac{2^{-0.01d}}{2}.
\]  \hspace{1cm} (8.3)

For \( \Phi \), using observation (i) and the fact that the central binomial coefficient is the largest one, we have

\[
\left( \frac{\frac{2d}{3} - \alpha d}{\frac{2d}{3} - \alpha d + c} \right) = \left( \frac{\frac{2d}{3} + c}{\frac{2d}{3} + c - \alpha d + 1} \right) \left( \frac{\frac{2d}{3} + c}{\frac{2d}{3} + c - \alpha d + 1} \right) \left( \frac{d}{\alpha d} \right)^{\alpha d} \left( \frac{2d}{\alpha d + c} \right) \left( \frac{2d}{3} - 1 + \alpha d \right)
\]

\[
\leq \left( \frac{\frac{2d}{3} - \alpha d + c}{\frac{2d}{3} - \alpha d + c} \right)^{\alpha d} \left( \frac{\frac{2d}{3} + c}{\alpha d - \alpha} \right)^{\alpha d}.
\]

Again using the bounds on \( \alpha, c \) and \( d \) we obtain

\[
\frac{\Phi}{(\frac{2d}{3^d})} \leq d^2 \left( \frac{2d}{3} + c \right) \left( \frac{\frac{2d}{3} + c}{\frac{2d}{3} - \alpha} \right)^{\alpha d} \leq 2^{0.043d} 2^{-0.058d} \leq \frac{2^{-0.01d}}{2}.
\]  \hspace{1cm} (8.4)

Adding the bounds from (8.3) and (8.4) concludes the proof of the lemma.
8.7 Concluding the proof of Theorems 8.3 and 8.4

Proof of Theorem 8.3 Without loss of generality assume $a = d/402$ and $\delta > 0$. Let $\alpha = 2^{0.009d}$ and let $M = \max_{f,x} \Pr_{\sigma} (|L(\sigma, f, x)| > \alpha)$. We claim that $M \geq 1/2 - 1.1\delta$, which then implies the theorem. For the sake of contradiction, suppose that $M < 1/2 - 1.1\delta$. Since then $\delta + M < 1/2$, we can take an integer $c \in (d/201, d/200)$ and employ Lemma 8.3 to get that there is a $c$-connector for $P$ with maximum outdegree at most $2d/\alpha \log(\frac{1}{2\delta + 2M}) \leq 2d\alpha / \log(\frac{1}{1-0.2\delta})$. Since $\delta > 0$ and $d$ is sufficiently large with respect to $\delta$, we obtain that this connector has maximum outdegree less than $2^{0.01d}$. This contradicts Theorem 8.5 and concludes the proof of Theorem 8.3.

Proof of Theorem 8.4 By definition, a 1-connector is also a $c$-connector for any $c \geq 1$. We proceed as in the proof of Theorem 8.3, but now with no restriction on $a$. \qed
Chapter 9

Application: filter mechanism for data privacy

This chapter is organized as follows. In Section 9.1, we formally define the filter mechanism and analyze its performance. In Section 9.2, we review differential privacy and the Laplace mechanism from [DMNS06]. In Section 9.3, we instantiate the filter mechanism with the Laplace mechanism and the filter from Theorem 7.1 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.

9.1 Filter mechanism

The following theorem summarizes the performance of the filter mechanism.

**Theorem 9.1** (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 $f : D \rightarrow R$, 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 2.5. 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 $\delta$ is the error probability of $F$.

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

Proof. Let $x$ be the input database and $g_\rho$ be the output function of the local Lipschitz filter $F$ with input function $f/c$ and random seed fixed to $\rho$. Since the filter is local, $g_\rho$ is well defined on $D$. In particular, this means that $g_\rho$ can be computed by the user without the knowledge of $x$ and therefore does not disclose anything about the database $x$. Moreover, $g_\rho$ 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 $\rho$ was arbitrary, this analysis holds for any choice of $\rho$, 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. \qed

9.2 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 $i$th component represents the number of times the $i$th element of $U$ occurs in the database. (Such representation is considered, e.g., in [HT10].) 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 9.1** (Differential privacy, [DMNS06]). Fix $\epsilon > 0$. A randomized algorithm $A$ is $\epsilon$-differentially private if for all neighbors $x, y \in D$, and for all subsets $S$ of outputs, $\Pr[A(x) \in S] \leq e^\epsilon \Pr[A(y) \in S]$.

Recall that $\text{Lap}(\lambda)$ denote the Laplace distribution on $\mathbb{R}$ with the scale parameter $\lambda$. The **Laplace mechanism** [DMNS06] is a randomized algorithm for evaluating functions on databases privately and efficiently.

**Theorem 9.2** (Laplace Mechanism [DMNS06]). 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 $\epsilon$-differentially private:

$$A_{\text{Lap}}^t(x) = f(x) + (Y_1, \ldots, Y_t),$$

where $Y_i \overset{i.i.d.}{\sim} \text{Lap}(c/\epsilon)$ 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.

### 9.3 Instantiating the filter mechanism for histograms

Theorem 9.1 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 7.1, 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 $i$th component
represents the number of times the $i$th 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 $H_{m+1,k}$ (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 7.1 in the filter mechanism to release functions $f : D \rightarrow \mathbb{R}$. The performance of the resulting algorithm is summarized in Corollary 9.1. We also bound the error of the mechanism. Given a function $f : D \rightarrow \mathbb{R}$ 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 9.1** (Filter mechanism for histograms). Fix $c, \epsilon > 0$. For all functions $f : D \rightarrow \mathbb{R}$, the filter mechanism of Theorem 9.1 instantiated with the Laplace mechanism and the local filter of Theorem 7.1 is $\epsilon$-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 $H_{m+1,k}$ are adjacent, it follows that the metric $d_D$ on $D$ is given by the shortest path metric on $H_{m+1,k}$. Therefore, using the local Lipschitz filter from Theorem 7.1 in the filter mechanism of Theorem 9.1 instantiated with the Laplace mechanism to release functions $f : D \rightarrow \mathbb{R}$, 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 $f(x) + \text{Lap}(c/\epsilon)$. This is because the local filter of Theorem 9.1 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 $\epsilon$-differentially private mechanisms for releasing functions $f : D \rightarrow \mathbb{R}$, 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 $H_{m+1,k}$, it can be done by exhaustive search over all edges of $H_{m+1,k}$ 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 \( f : D \rightarrow \mathbb{R} \) 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 \( B(x) = f \circ A_{\text{Lap}}^{\text{identity}}(x) \) denote the resulting mechanism. In Theorem 9.3, 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.

**Theorem 9.3.** There exist functions \( f \) such that releasing \( f \) results in expected error \( \Omega(\sqrt{k}/\epsilon) \) with the noisy histogram approach, but only \( O(1/\epsilon) \) with the filter mechanism.

**Proof.** Given \( S \subseteq \{0, \ldots, m\}^k \), let \( f_S : D \rightarrow \mathbb{R} \) be the function which on input \( x \in D = \{0, \ldots, m\}^k \), 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, \( E(f_S, B) \), is \( \Omega(\sqrt{k}/\epsilon) \). In contrast, for each \( S \subseteq \{0, \ldots, m\} \), the error of the filter mechanism, \( E(f_S, A_{\text{Fil}}) \), is \( O(1/\epsilon) \), by Corollary 9.1 and the fact that \( f_S \) is Lipschitz for each \( S \subseteq \{0, \ldots, m\} \).

On query \( f_S \) and database \( x \), the noisy histogram approach outputs
\[
 B(x) = \sum_{i \in [r]} (x_i + Y_i),
\]
where \( Y_i \)'s are independently and identically distributed random variables in \( \text{Lap}(1/\epsilon) \) and \( |S| = r \). Denoting by \( Z_r \) the random variable \( Y_1 + \ldots + Y_r \), we see that 
\[
 E(f_S, B) = \sup_{x \in D} E[|Z_r|].
\]
Let \( \text{Bad} \) denote the event \( |Z_r| > \sqrt{r}/\epsilon \). Then, 
\[
 E[|Z_r|] \geq E[|Z_r| \mid \text{Bad}] \cdot Pr[\text{Bad}] \geq (\sqrt{r}/\epsilon) \cdot Pr[\text{Bad}].
\]
Since \( r = \Omega(k) \), it suffices to show that \( \text{Bad} \) occurs with constant probability. Towards this, let \( b = 1/\epsilon \). Now, 
\[
 E[Z_r^4] = 12b^4r(r + 1) \leq 24b^4r^2 = 6(E[Z_r^2])^2.
\]
The inequality holds because (we may assume) \( r \) is at least 1 while the last equality holds because \( E[Z_r^2] = 2b^2r \). Since, \( Z_r \) is symmetric about 0, using anti-concentration results of [GOWZ10], restated below as Claim 9.1, 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 9.3.
Claim 9.1 (Fact 3.3, Proposition 3.7, [GOWZ10]). 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 $0 < t < 1$, $\Pr[|X - \theta| > t \mathbb{E}[X^2]^{1/2}] \geq \eta^4(1 - t^2)^2$, where $\eta = \min(\eta', 1/\sqrt{3})$. 
Chapter 10

Application: testing privacy

In this chapter, we present application of Lipschitz testers to testing privacy. Section 10.1 describes the connection between differential privacy and testing the Lipschitz property. The same section defines the privacy notion that we test: differential privacy on typical datasets (DPTD). In Section 10.2, we give algorithms for testing DPTD. Finally, in Section 10.3, we show how to use a DPTD tester to convert an arbitrary algorithm to an algorithm which always satisfies DPTD. The transformation does not change the output distribution of a differentially private algorithm.

10.1 Differential privacy and its connections to testing the Lipschitz property

In Chapter 9, we reviewed the definition of differential privacy from [DMNS06]. In this chapter, we look at a generalization of this definition from [DKMN06]. Also, in this chapter, we focus on datasets which are fixed-length vectors from an arbitrary finite domain $\mathcal{X}^d$, where each coordinate represents one person’s data. (For example, when $\mathcal{X}^d$ is $\{0, 1\}^d$, datasets consist of $d$-bit vectors and each person’s data is a Boolean value.) Recall that, intuitively, the output of a differentially private algorithm is almost the same whether or not a specific person’s data is present in the dataset. More precisely, an algorithm is differentially private if it has similar output distributions on input datasets which are close in Hamming distance. The Hamming distance between $x$ and $x'$, denoted $d_H(x, x')$, is the number of coordinates on which $x$ and $x'$ differ.
Definition 10.1 ((\(\alpha, \beta\))-Differential privacy [DMNS06, DKMN06]). A randomized algorithm \(A\) is \((\alpha, \beta)\)-differentially private if for any two datasets \(x\) and \(x'\) in \(\mathcal{X}^d\), and for all measurable sets \(Z \subseteq \text{Range}(A)\), the following holds:

\[
\Pr[A(x) \in Z] \leq e^{\alpha d_H(x, x')} \Pr[A(x') \in Z] + \beta. \tag{10.1}
\]

If \(\beta = 0\), algorithm \(A\) is called \(\alpha\)-differentially private.

In this chapter, we focus on differentially private algorithms which output values in a finite range space \(Z\). Such algorithms have a clean characterization in terms of the Lipschitz property. Specifically, define functions \(f_z : x \rightarrow \mathbb{R}\) for all \(z \in Z\) by setting \(f_z(x) = \log \Pr[A(x) = z]\). We make the following simple but important observation.

Observation 10.1 (Differential privacy as a Lipschitz condition). Algorithm \(A\) is \(\alpha\)-differentially private if and only if for every \(z \in Z\), function \(f_z\) is \(\alpha\)-Lipschitz.

Therefore, one could check if an algorithm specified by functions \(f_z\) is differentially private, given oracle access to these functions, if one could design a procedure that decides if an input function is Lipschitz. However, as noted in the introduction of the thesis, deciding if a given function is Lipschitz is NP-hard. Can we still efficiently check for some relaxation of differential privacy? We consider relaxations of differential privacy which are based on distributional assumptions. Specifically, we adapt a relaxation from [BBG+11a] and show that it can indeed be tested using a connection to testing the Lipschitz property. The relaxation we consider assumes that datasets come from some fixed distribution \(\Pi\) on the set of all datasets. The notion of privacy is relaxed from the worst case guarantee over all pairs of datasets (i.e., differential privacy) to a notion where the differential privacy condition is required to hold only on datasets which are more likely to occur (i.e., have high-probability mass under distribution \(\Pi\)). We refer to this notion as differential privacy on typical datasets (DPTD) (Definition 10.2). As mentioned earlier, DPTD is an adaptation of more general definition introduced in [BBG+11a] under the name of generalized differential privacy (GDP). GDP was defined in the context of a related distributional notion of privacy called noiseless privacy, first introduced by [BBG+11b]. A related notion called natural differential privacy has also been recently proposed by [BD12]. Since we focus on DPTD in this work, we do not discuss noiseless privacy (and its variants) further. Next we give a formal definition of
DPTD. The definition is parametrized by three parameters $\alpha$, $\beta$ and $\gamma$. The parameters $\alpha$ and $\beta$ play the same role as in the differential privacy definition, while the parameter $\gamma$ bounds the probability of the “bad” set $B$ of datasets on which the differential privacy condition fails to hold.

**Definition 10.2** ($(\alpha, \beta, \gamma)$-DPTD). Let $\Pi$ be a fixed distribution on the domain $X^d$ of datasets. A randomized algorithm $\mathcal{A}$ is $(\alpha, \beta, \gamma)$-differentially private on typical datasets (DPTD) if there exists a subset $B \subseteq X^d$ satisfying $\Pr_{x \sim \Pi}[x \in B] \leq \gamma$, such that condition (10.1) of Definition 10.1 holds for any two datasets $x, x' \in X^d \setminus B$ and all measurable sets $Z \subseteq \text{Range}(\mathcal{A})$. The probability in (10.1) is over the randomness of the algorithm $\mathcal{A}$.

Our main observation is that for algorithms which output values in a finite range, testing DPTD can be reduced to testing the Lipschitz property (of a family of functions).

**Observation 10.2** (DPTD for algorithms with finite range). Algorithm $\mathcal{A}$ is $(\alpha, 0, \gamma)$-DPTD if and only if the following two conditions hold: (i) there exists a subset $B \subseteq X^d$ such that $\Pr_{x \sim \Pi}[x \in B] \leq \gamma$; and (ii) for every $z \in Z$, function $f_z$ is $\alpha$-Lipschitz on the set $X^d \setminus B$.

Recall that a function $f$ is $\epsilon$-close to property $\mathcal{P}$ with respect to distribution $\Pi$ if there is a function $g$ which satisfies $\mathcal{P}$ and $\Pr_{x \sim \Pi}[f(x) \neq g(x)] \leq \epsilon$. Observation 10.2, in particular, implies the following. If algorithm $\mathcal{A}$ satisfies $(\alpha, 0, \gamma)$-DPTD, then for every $z \in Z$, function $f_z$ is $\gamma$-close to the $\alpha$-Lipschitz property with respect to the distribution $\Pi$. However, to apply a Lipschitz tester, we need the converse of this statement. The following lemma gives a statement in the spirit of the converse.

**Lemma 10.1** (Connection between DPTD and testing the Lipschitz property). If for every $z \in R$, function $f_z$ is $\epsilon_z$-close to the $\alpha$-Lipschitz property with respect to the distribution $\Pi$, then $\mathcal{A}$ is $(\alpha, 0, \sum_z \epsilon_z)$-DPTD. In particular, if $\mathcal{A}$ is not $(\alpha, 0, \gamma)$-DPTD, then there exists $z \in Z$ such that $f_z$ is $\gamma / |Z|$-far from the $\alpha$-Lipschitz property.

**Proof.** Since every $f_z$ is $\epsilon_z$-close to the $\alpha$-Lipschitz property with respect to the distribution $\Pi$, there exists $B_z$ corresponding to each $f_z$ such that (i) $f_z$ is $\alpha$-Lipschitz on
\( \mathcal{X}^d \setminus B_z \); and (ii) \( \Pr_{x \sim \Pi}[x \in B_z] \leq \epsilon_z \). Let \( B \) be the union over all \( z \) of the sets \( B_z \). Applying the union bound, we get \( \Pr_{x \sim \Pi}[x \in B] \leq \sum_z \epsilon_z \). Then the first part of the lemma follows from Observation 10.2 with \( B \) as the required set. The second part of the lemma follows from an averaging argument.

10.1.1 Discussion of differential privacy on typical datasets

In Definition 10.2 of DPTD, if we assume \( \beta = 0 \) for simplicity, then, at a high level, DPTD implies that for any two datasets \( x \) and \( x' \) from the set \( \mathcal{X}^d \setminus B \), which have sufficient probability mass under \( \Pi \) and differ in \( k \) entries, the distribution of \( A(x) \) and \( A(x') \) have a statistical distance of \( 2k\alpha \) when \( k\alpha \) is less than one.

In differential privacy, the scale of the parameters \( \alpha \) and \( \beta \) are typically chosen as follows: \( \alpha \) is chosen to be some small constant and \( \beta \) is chosen to be \( O(1/d^2) \). With this choice of parameters, differential privacy ensures that even in the presence of any auxiliary information, from the output of the algorithm \( A \), an adversary draws the same conclusions about any entry in the data set irrespective of its presence or absence. (See [KS08] for more discussion.) Since \( \alpha \) and \( \beta \) play the same role in DPTD, we think of \( \alpha \) and \( \beta \) as being of the same order as discussed for differential privacy. Additionally, throughout this paper we think of \( \gamma \) as being of the same order as \( \beta \).

In [BBG+11a], a generalization of DPTD has been stated under the name of generalized differential privacy (GDP). Under suitable choice of auxiliary information (i.e., the random variable \( A_{ux} \)), the definition of [BBG+11a] reduces to Definition 10.2. More precisely, for every data entry \( x_i \), the auxiliary information in the GDP condition (of the definition of GDP) corresponds to all entries in the data set \( x \) except \( x_i \).

10.2 Testing differential privacy on typical datasets

In this section, we formally define the notion of a DPTD tester. Next we describe our main result (Theorem 10.1) showing a reduction from DPTD testing to Lipschitz testing. This reduction uses an arbitrary Lipschitz tester as a black box and is presented in Section 10.2.1. Later, in Section 10.2.2, we show a more efficient transformation of privacy testing to Lipschitz testing. This transformation requires a particular type of Lipschitz testers which we call proximity-proportional Lipschitz testers.
Definition 10.3 (DPTD tester). Consider a randomized algorithm $A$ which takes a dataset $x \in \mathcal{X}^d$ as input and outputs a value in the finite set $\mathcal{Z}$. Define functions $f_z : x \rightarrow \mathbb{R}$ for every $z \in \mathcal{Z}$ by setting $f_z(x) = \log \Pr[A(x) = z]$. Let $\Pi$ be a fixed but unknown distribution on datasets $\mathcal{X}^d$. Fix $\alpha > 0$ and $\beta, \gamma, \delta \in (0, 1]$. An $(\alpha, \beta, \gamma, \delta)$-DPTD tester with respect to distribution $\Pi$ on $\mathcal{X}^d$ is a randomized algorithm (denoted $T_{priv}$) that gets oracle access to functions $f_z$'s (for every $z \in \mathcal{Z}$) and oracle access to independent samples from distribution $\Pi$. We refer to this as running the DPTD tester on input $A$. The algorithm satisfies the following guarantees.

1. If $A$ is $\alpha$-differentially private, then $T_{priv}$ accepts.

2. If $A$ is not $(\alpha(1 + \delta), 0, \gamma)$-DPTD with respect to distribution $\Pi$, then $T_{priv}$ rejects with probability at least $1 - \beta$.

We call $\alpha$ and $\gamma$ the privacy parameters, $\beta$ the failure probability and, finally, $\delta$ the approximation parameter.

10.2.1 Testing DPTD using arbitrary Lipschitz tester

In this section, we prove the following theorem. Recall the definition of DPTD-tester and (approximate) Lipschitz tester given in Definitions 10.3 and 2.3, respectively.

Theorem 10.1 (DPTD tester based on approximate Lipschitz tester). Let $T_{Lip}$ be an approximate Lipschitz tester for testing the Lipschitz property of functions $f : \mathcal{X}^d \rightarrow \mathbb{R}$ with respect to distribution $\Pi$ on $\mathcal{X}^d$. Assume $T_{Lip}$ has 1-sided error and runs in time $t(|\mathcal{X}|, d, \epsilon, \delta)$, where $\epsilon \in (0, 1)$ is the proximity parameter and $\delta \in (0, 1)$ is the approximation parameter. (See Definition 2.3.) There exists an $(\alpha, \beta, \gamma, \delta)$-DPTD tester (given in Algorithm 7) with respect to distribution $\Pi$ on $\mathcal{X}^d$, which uses $T_{Lip}$ as a black box and runs in time $O(|\mathcal{Z}| \cdot t(|\mathcal{X}|, d, \gamma / |\mathcal{Z}|, \delta) \cdot \log 1 / \beta)$.

Proof of Theorem 10.1 The following algorithm gives the DPTD tester stated in Theorem 10.1.
Algorithm 7: DPTD tester based on arbitrary Lipschitz tester

**Input**: Oracle access to values $\Pr[A(x) = z]$ for all $x \in \mathcal{X}^d$ and $z \in \mathcal{Z}$, oracle access to independent samples from the (unknown) product distribution $\Pi$ on $\mathcal{X}^d$, domain parameters $d$ and $|\mathcal{X}|$, range parameter $\mathcal{Z}$, proximity parameter $\epsilon$, privacy parameters $\alpha, \gamma$, failure probability $\beta$, and approximation parameter $\delta$.

1. Let $T_{Lip}$ be a 1-sided error $(1 + \delta)$-approximate Lipschitz tester for functions $f : \mathcal{X}^d \rightarrow \mathbb{R}$ (see Definition 2.3).

2. For all $z \in \mathcal{Z}$ do
   3. Define function $f_z : \mathcal{X}^d \rightarrow \mathbb{R}$ by setting $f_z(x) = \frac{1}{\alpha} \log \Pr(A(x) = z)$.
   4. Repeat $\log(1/\beta)$ times independently:
      5. Run $T_{Lip}$ on function $f_z$ with proximity parameter $\gamma/|\mathcal{Z}|$ and approximation parameter $\delta$.
      6. If $T_{Lip}$ rejects, then reject.
   7. Accept.

We use Lemma 10.1 and Observation 10.1 to prove the theorem. To show that $T_{priv}$ satisfies the first item of Definition 10.3, assume $A$ is $\alpha$-differentially private. Then Observation 10.1 implies that for every $z \in \mathcal{Z}$, function $f_z$ (defined in Step 2 of Algorithm 7) is Lipschitz. Since $T_{Lip}$ always accepts a Lipschitz function, we get that $T_{priv}$ accepts, as required. To show that $T_{priv}$ satisfies the second item of Definition 10.3, assume $A$ is not $(\alpha(1 + \delta), 0, \gamma)$-DPTD. Then Lemma 10.1 implies that there exists $z^*$ such that $f_{z^*}$ is $\gamma/|\mathcal{Z}|$-far from $(1 + \delta)$-Lipschitz. Since $T_{Lip}$ is a $(1 + \delta)$-approximate Lipschitz tester, it follows that $T_{Lip}$ rejects $f_{z^*}$ with probability at least $2/3$, in each independent invocation of Steps 5-6. Since Steps 5-6 is repeated $\log(1/\beta)$ times independently, it follows from standard arguments that function $f_{z^*}$ gets rejected with probability at least $1 - \beta$, as required. The running time of $T_{priv}$ follows from the fact that the tester $T_{Lip}$ is invoked at most $|\mathcal{Z}| \cdot \log(1/\beta)$ times.

When each row of database is Boolean-valued (i.e., $|\mathcal{X}| = 2$), we can instantiate Theorem 10.1 with the $(1 + \delta)$-approximate Lipschitz tester developed in Chapter 4 (see Corollary 4.1) to obtain the following.

**Corollary 10.1** (DPTD tester for product distributions). There exists an $(\alpha, \beta, \gamma, \delta)$-DPTD tester with respect to an arbitrary product distribution on $\{0, 1\}^d$ which runs in
time $O(|Z|^2d^2 \log(1/\beta)/\delta \gamma)$ assuming $\delta \leq \gamma/(2|Z|d^2)$, where $Z$ is the range of the algorithm being tested.

Chakrabarty and Seshadhri [CS13b] give a Lipschitz tester for functions on $\{0, 1\}^d$ with respect to the uniform distribution on the domain. It runs in time $O(d/\epsilon)$. Instantiating Theorem 10.1 with this tester gives a DPTD-tester with respect to the uniform distribution. (Since the tester is a standard tester, $\delta = 0$ in the following.)

**Corollary 10.2** (DPTD tester for uniform distribution). *There is an $(\alpha, \beta, \gamma, 0)$-DPTD tester with respect to the uniform distribution on $\{0, 1\}^d$. The tester has running time $O(|Z|^2d^2 \log(1/\beta)/\gamma)$, where $Z$ is the range of the algorithm being tested.*

### 10.2.2 Faster DPTD tester using proximity-proportional Lipschitz tester

The proof of Theorem 10.1 relied on the second statement from Lemma 10.1 and used an *arbitrary* Lipschitz tester. It is possible to obtain a faster privacy tester by relying on the first (stronger) statement from Lemma 10.1. This requires making a mild assumption about the guarantees of the Lipschitz tester. Specifically, we consider *proximity oblivious* testers defined by Goldreich and Ron [GR11]. These are testers whose rejection probability is a function of solely the input’s actual distance to the property. We consider the extension (which was also noted in [GR11]) where the rejection probability is allowed to depend on other parameters of the input. However, we impose the stronger requirement that the rejection probability be *linear* in the distance of the input to the property. Given a function $f : X^d \to \mathbb{R}$ and a distribution $\Pi$ on $X^d$, recall from Definition 2.1 that $\epsilon_f$ denotes the distance of $f$ to the Lipschitz property with respect to the distribution $\Pi$.

**Definition 10.4** (Proximity-proportional tester). *A tester is proximity-proportional if it satisfies Item 1 of Definition 2.2 and the following condition: If $f$ does not satisfy property $\mathcal{P}$, then the tester rejects with probability at least $c \cdot \epsilon_f$, where “constant” $c$ may depend on domain parameters (such as $|X|$ and $d$ for functions on $X^d$) but is independent of $\epsilon_f$, where $\epsilon_f$ is as defined in Definition 2.1.*

**Theorem 10.2** (DPTD tester based on proximity-proportional Lipschitz tester). *Let $\mathcal{T}_{obl}$ be a proximity-proportional tester (as defined in Definition 10.4) for testing the Lipschitz
property of functions $f : \mathcal{X}^d \to \mathbb{R}$ with respect to distribution $\Pi$ on $\mathcal{X}^d$. Suppose it rejects a non-Lipschitz function with probability at least $c(|\mathcal{X}|, d) \cdot \epsilon_f$ and runs in time $t(|\mathcal{X}|, d)$. There exists an $(\alpha, \beta, \gamma, \delta)$-DPTD tester (specified in Algorithm 8) with respect to distribution $\Pi$ on $\mathcal{X}^d$, which uses $\mathcal{T}_{obl}$ as a black box and runs in time

$$O \left( \frac{|\mathcal{Z}|}{c(|\mathcal{X}|, d) \cdot \gamma} \cdot t(|\mathcal{X}|, d) \cdot \log 1/\beta \right).$$

**Proof of Theorem 10.2** The following algorithm gives the DPTD tester stated in Theorem 10.2.

**Algorithm 8: DPTD tester based on proximity-proportional Lipschitz tester**

**Input**: Same as that of Algorithm 7 except the approximation parameter $\delta$ is not an input.

1. Let $\mathcal{T}_{obl}$ be the tester in the statement of Theorem 10.2.
2. Repeat $\log(1/\beta) \cdot \frac{|\mathcal{Z}|}{c(|\mathcal{X}|, d) \cdot \gamma}$ times:
   3. Pick $z$ uniformly at random from $\mathcal{Z}$.
   4. Define function $f_z : \mathcal{X}^d \to \mathbb{R}$ by setting $f_z(x) = \frac{1}{\alpha} \log \Pr(A(x) = z)$.
   5. Run $\mathcal{T}_{obl}$ on $f_z$ and reject if $\mathcal{T}_{obl}$ rejects.
   6. Accept.

If $A$ is $\alpha$-differentially private, then every $f_z$ is Lipschitz. Since $\mathcal{T}_{obl}$ is a 1-sided error tester (by definition), it always accepts a Lipschitz function. Therefore, every execution of Step 5 accepts and so does Algorithm 8. Next consider the case when $A$ is not $(\alpha, 0, \gamma)$-DPTD. Then, by Lemma 10.1, $\sum_z \epsilon_{f_z} > \gamma$. We show that the probability that a single iteration of the repeat loop of Algorithm 8 rejects is at least $(c(|\mathcal{X}|, d) \cdot \gamma)/|\mathcal{Z}|$. The fact that Algorithm 8 rejects with probability $1 - \beta$ then follows from standard arguments. We therefore analyze a single iteration of the repeat loop:

$$\Pr[\text{Algorithm 8 rejects in a single execution of Steps 3-5}]$$

$$= \frac{1}{|\mathcal{Z}|} \cdot \sum_{z \in \mathcal{Z}} \Pr[\mathcal{T}_{obl} \text{ rejects } f_z \text{ in a single execution of Step 5}]$$

$$\geq \frac{1}{|\mathcal{Z}|} \cdot \sum_{z \in \mathcal{Z}} c(|\mathcal{X}|, d) \cdot \epsilon_{f_z} = \frac{c(|\mathcal{X}|, d)}{|\mathcal{Z}|} \cdot \sum_{z \in \mathcal{Z}} \epsilon_{f_z} \geq \frac{c(|\mathcal{X}|, d)}{|\mathcal{Z}|} \cdot \gamma.$$
The first inequality holds because $T_{obl}$ is a proximity-proportional Lipschitz tester. The second inequality follows from the observation made earlier: $\sum z \epsilon_z > \gamma$. □

The following corollary gives an example instantiation of Theorem 10.2. It is based on the proximity-proportional tester for testing the Lipschitz property of functions on $\{0, 1\}^d$ with respect to the uniform distribution, given by Chakrabarty and Seshadhri in [CS13b]. This tester runs in constant time and rejects a non-Lipschitz function with probability $\Omega(\epsilon_f/d)$.

**Corollary 10.3** (DPTD tester for uniform distribution). There is an $(\alpha, \beta, \gamma, 0)$-DPTD tester with respect to arbitrary uniform distribution on $\{0, 1\}^d$ which uses a proximity-proportional Lipschitz tester from [CS13b] as a black box. The tester has running time $O(|Z|d \log(1/\beta)/\gamma)$, where $Z$ is the range of the algorithm being tested.

Observe we save a factor of $|Z|$ in the running time when compared to using standard Lipschitz testers (see Corollary 10.2).

### 10.3 Transforming an arbitrary algorithm to its DPTD variant

In this section, we state and prove Theorem 10.3 which gives a way of converting an arbitrary input algorithm $A$ to a related algorithm $A'$ which always satisfies DPTD (even if $A$ does not). Moreover, the output distributions of $A$ and $A'$ are identical if $A$ is differentially private.

**Theorem 10.3** (DPTD mechanism). Fix parameter $\alpha > 0$ and parameters $\beta, \gamma, \delta \in (0, 1]$. Let $A$ be an arbitrary randomized algorithm (purported to be differentially private) which takes as input a dataset $x \in X^d$ and outputs a value in the finite set $Z$. Let $A'$ be an algorithm with oracle access to $A$ (i.e., it can get $A(x)$ for each $x \in X^d$). Moreover, assume it has oracle access to values $\Pr[A(x) = z]$ for every $x \in X^d$ and $z \in Z$. Define $A'$ so that on input dataset $x \in X^d$, it runs an $(\alpha, \beta, \gamma, \delta)$-DPTD tester on input $A$ and outputs $A(x)$ if and only if the tester accepts. If the tester rejects, it outputs fail. Then the following holds.

- **(privacy)** Algorithm $A'$ is $(\alpha(1 + \delta), \beta, \gamma)$-DPTD.
• (utility) If the input algorithm $A$ is $\alpha$-differentially private, then the output distributions of algorithms $A'$ and $A$ are identical.

Proof. Let $T_{\text{priv}}$ be the DPTD tester stated in the theorem statement. If the input algorithm $A$ is $\alpha$-differentially private, then by Theorem 10.1, $T_{\text{priv}}$ always accepts. Therefore, in this case, $A'(x)$ is identical to $A(x)$. The claim about the utility, namely, the second part of the theorem follows. Next we prove Algorithm $A'$ is $(\alpha(1 + \delta), \beta, \gamma)$-DPTD.

The output space of algorithm $A'$ is $\mathcal{Z} \cup \{\text{fail}\}$. For all datasets $x$ and all $z \in \mathcal{Z}$, the following holds:

$$
\Pr[A'(x) = z] = \Pr[A'(x) = z|T_{\text{priv}} \text{ accepts } A] \cdot \Pr[T_{\text{priv}} \text{ accepts } A] = \Pr[A(x) = z] \cdot \Pr[T_{\text{priv}} \text{ accepts } A].
$$

(10.2)

Also, for every dataset $x$, we have the following.

$$
\Pr[A'(x) = \text{fail}] = \Pr[T_{\text{priv}} \text{ rejects } A]
$$

(10.3)

We break down the proof into two cases depending on whether the input algorithm $A$ is $(\alpha(1 + \delta), 0, \gamma)$-DPTD or not.

**Case 1:** Assume $A$ is $(\alpha(1 + \delta), 0, \gamma)$-DPTD. Then there exists a set $B \subseteq \mathcal{X}^d$ satisfying $\Pr_{x \sim \Pi}[x \in B] \leq \gamma$ such that for any two datasets $x, x' \in \mathcal{X}^d \setminus \{0, 1\}$ and any $z \in \mathcal{Z}$, the following holds.

$$
\Pr[A(x) = z] \leq e^{\alpha(1+\delta)d_H(x,x')} \Pr[A(x') = z]
$$

Multiplying both sides of the above equation by $\Pr[T_{\text{priv}} \text{ accepts } A]$ and using (10.2) on both sides of the resulting inequality, we get that the following holds for all $x, x'$ and $z$ as above.

$$
\Pr[A'(x) = z] = \Pr[A(x) = z] \cdot \Pr[T_{\text{priv}} \text{ accepts } A] \\
\leq e^{\alpha(1+\delta)d_H(x,x')} \Pr[A(x') = z] \cdot \Pr[T_{\text{priv}} \text{ accepts } A] \\
= e^{\alpha(1+\delta)d_H(x,x')} \Pr[A'(x') = z]
$$
Also, (10.3) implies the following for all \( x, x' \in \mathcal{X}^d \) (and, in particular, for all \( x, x' \in \mathcal{X}^d \setminus B \)):

\[
\Pr[\mathcal{A}'(x) = \text{fail}] = \Pr[\mathcal{A}'(x') = \text{fail}].
\]

Therefore, \( \mathcal{A}' \) satisfies \((\alpha (1 + \delta), 0, \gamma)\)-DPTD with \( B \) as the required set.

**Case 2:** Assume \( \mathcal{A} \) is not \((\alpha (1 + \delta), 0, \gamma)\)-DPTD. Then for every \( z \in \mathcal{Z} \), we get the following, where the last inequality holds for arbitrary dataset \( x' \).

\[
\Pr[\mathcal{A}'(x) = z] = \Pr[\mathcal{A}(x) = z] \cdot \Pr[\mathcal{T}_{\text{priv}} \text{ accepts } \mathcal{A}] 
\leq \Pr[\mathcal{T}_{\text{priv}} \text{ accepts } \mathcal{A}] 
\leq \beta 
\leq \beta + e^{\alpha (1+\delta)d_H(x,x')} \Pr[\mathcal{A}'(x') = z]
\]

Again, (10.3) implies the following for all \( x, x' \in \mathcal{X}^d \).

\[
\Pr[\mathcal{A}'(x) = \text{fail}] = \Pr[\mathcal{A}'(x') = \text{fail}].
\]

Therefore, in this case, \( \mathcal{A}' \) satisfies \((\alpha (1 + \delta), \beta, 0)\)-DPTD with \( B = \emptyset \) as the required set.

Thus, we see that in either cases, \( \mathcal{A}' \) satisfies \((\alpha (1+\delta), \beta, \gamma)\)-DPTD, as required. \( \square \)
Chapter 11

Future work

We list a few possible directions for the future work.

- It would be interesting to pursue various relaxations of the definition of the Lipschitz filter to circumvent the lower bounds of Chapter 8. Specifically, the requirement that the filter output a Lipschitz function may be relaxed to only require that the filter output a $\kappa$-Lipschitz function for $\kappa > 1$, and furthermore, only *with high probability*. Another relaxation could be to require that the filter be accurate only when input function $f$ belongs to a particular class $\mathcal{H}$. These relaxations still yield useful filter mechanisms (described in Chapter 9) and hence will have applications to data privacy.

- Property testing has its roots in program checking [BLR93, RS96]. Still the power of property testing in certifying properties of programs is somewhat underutilized. It would be interesting to combine code-based program analysis and black box sublinear algorithms (such as those developed in this thesis) to enable more advanced certification of properties of programs.

- In the privacy application of Chapter 10, we assumed that the algorithm $\mathcal{A}$ being tested has a finite discrete range space $\mathcal{Z}$. Furthermore, we assumed access to values $\Pr[\mathcal{A}(x) = z]$ for every $z \in \mathcal{Z}$. Removing these restrictions (e.g., allowing $\mathcal{Z}$ to be a continuous range space, such as $\mathbb{R}$) is an important open problem. This also raises some interesting new challenges in property testing of more general interest. For example, can we do property testing with noisy oracles (which only
approximately output the value of the function being tested)? Similarly, can we do property testing over continuous domains such as $\mathbb{R}$? Finally, it would be interesting to see if more stringent notions of privacy can be tested.
Bibliography


Vita

Madhav Jha

Madhav Jha is a PhD candidate in the Department of Computer Science and Engineering at the Pennsylvania State University. Madhav started his PhD program in 2008 after completing his Bachelor of Technology in Computer Engineering in 2008 from Institute of Technology, Banaras Hindu University (IT-BHU), Varanasi, India. Madhav’s current interests are in the areas of sublinear algorithms and data privacy. His research articles have been published in conferences such as FOCS, RANDOM, TCC and KDD, as well as journals such as SIDMA and SICOMP.