BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

ANALYZING MASSIVE DATASETS WITH MISSING ENTRIES: MODELS AND ALGORITHMS

by

NITHIN VARMA

B.Tech., National Institute of Technology Calicut, India, 2011
M.Sc., Tata Institute of Fundamental Research Mumbai, India, 2014

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2019

Approved by

First Reader

Sofya Raskhodnikova, PhD
Professor, Computer Science

Second Reader

Ronitt Rubinfeld, PhD
Professor, EECS MIT

Third Reader

Adam Smith, PhD
Professor, Computer Science

കാണാതെ പോയതൊക്കെയും

കാട്ടിത്തന്നീടും അംബികേ

കുട്ടികൾക്കൊക്കെയും നാഥേ

കൊട്ടിഞ്ചിരി നമോസ്തുതേ.

പണ്ഡിതരാജൻ കൊച്ചിക്കാവ് തമ്പുരാട്ടി

1882 - 1963

കൊടുങ്ങല്ലൂർ കോവിലകം

You reveal Truths hidden,

Goddess to us children!

Princess Kochikkavu, paṇḍitarājan

1882 - 1963

Royal House of Kodungallur

ACKNOWLEDGMENTS

First and foremost, I want to thank my advisor Sofya Raskhodnikova for all of her support and encouragement throughout these past five years. Meetings with Sofya always brought clarity to my thoughts and questions about research. Her untiring mentorship in communication, networking, and writing has had a tremendous positive impact on me. She continues to inspire me in every aspect of research. I hope I can live up to the standards and principles that Sofya maintains so effortlessly.

I thank my committee members Steven Homer, Ronitt Rubinfeld, and Adam Smith for their support and invaluable feedback on this thesis.

The work included in this thesis was done in collaboration with Kashyap Dixit, Iden Kalemaj, Amit Levi, Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, Noga Ron-Zewi, Abhradeep Thakurta, and Yuichi Yoshida. In particular, Chapter 3 and Section 6.2 are co-authored with Kashyap, Sofya, and Abhradeep. Chapter 4 is co-authored with Iden and Sofya. Chapter 5 and Section 6.3 are co-authored with Sofya and Noga. Chapter 7 is co-authored with Amit, Ramesh, and Sofya. Chapter 8 is co-authored with Yuichi.

I would like to express my gratitude to Kobi Nissim for a suggestion that led to our work in Chapter 4, and Venkatesan Guruswami for helping to tighten the analysis of our local list erasure-decoder for the Hadamard code (Section 5.3) and also for making a suggestion that led to Observation 5.1.6. I am grateful to Prahladh Harsha, Or Meir, Ramesh Krishnan S. Pallavoor, Adam Smith, Sergey Yekhanin, and Avi Wigderson for discussions that were helpful to the work included in Chapters 5 and 6.

I am indebted to all my other collaborators on research projects not included in this thesis and all colleagues at IISc Bangalore, TIFR Mumbai, Pennsylvania State

University, Boston University, and National Institute of Informatics Tokyo. I thank you all for working with me, teaching me several lessons in research, answering my naive questions patiently, and for all the fun conversations that we have had in our meetings.

I was fortunate to work closely with Paul Medvedev and Yuichi Yoshida during my PhD I would like to thank Paul for his support during one of the semesters of my PhD I would also like to express my gratitude to Yuichi for hosting me as an intern at the National Institute of Informatics in the last semester of my PhD

I take this opportunity to thank the sources of support during my PhD years: (1) Pennsylvania State University Graduate Fellowship, (2) Pennsylvania State University College of Engineering Fellowship, (3) National Science Foundation grants CAREER (CCF-0845701), CCF-1422975 and CCF-1832228 awarded to Sofya Raskhodnikova, (4) National Science Foundation grant IIS-1453527 awarded to Paul Medvedev, (5) and National Institute of Informatics International Internship Program.

I extend my gratitude to the departmental staff at both Pennsylvania State University and Boston University for their help with all of the administrative formalities.

I consider myself blessed to have taken this PhD journey alongside two wonderful people – Om Thakkar and Ramesh Krishnan. Om and Ramesh were always there for me, in both professional and personal life. I thank them for being the good-hearted humans that they are.

Last but not least, I thank my family, friends, and teachers for their unconditional and unwavering care, love, and encouragement.

ANALYZING MASSIVE DATASETS WITH MISSING ENTRIES: MODELS AND ALGORITHMS

NITHIN VARMA

Boston University, Graduate School of Arts and Sciences, 2019

Thesis Advisor: Sofya Raskhodnikova, Professor of Computer Science

ABSTRACT

We initiate a systematic study of computational models to analyze algorithms for massive datasets with missing or *erased* entries and study the relationship of our models with existing algorithmic models for large datasets.

We focus on algorithms whose inputs are naturally represented as functions, codewords, or graphs. First, we generalize the property testing model (Rubinfeld & Sudan (1996); Goldreich, Goldwasser, & Ron (1998)), one of the most widely studied models of sublinear-time algorithms, to account for the presence of adversarially erased function values. We design efficient erasure-resilient property testing algorithms for several fundamental properties of real-valued functions such as monotonicity, Lipschitz property, convexity, and linearity.

We then investigate the problems of local decoding (Katz & Trevisan (2000)) and local list decoding (Sudan, Trevisan, & Vadhan (2001)) of codewords containing erasures. We show that, in some cases, these problems are strictly easier than the corresponding problems of decoding codewords containing errors. Moreover, we use this understanding to show a separation between our erasure-resilient property testing model and the (error) tolerant property testing model (Parnas, Ron, & Rubinfeld (2006)). The philosophical message of this separation is that errors occurring in large datasets are, in general, harder to deal with, than erasures.

Finally, we develop models and notions to reason about algorithms that are intended to run on large graphs with missing edges. While running algorithms on large graphs containing several missing edges, it is desirable to output solutions that are close to the solutions output when there are no missing edges. With this motivation, we define average sensitivity, a robustness metric for graph algorithms. We discuss various useful features of our definition and design approximation algorithms with good average sensitivity bounds for several optimization problems on graphs. We also define a model of erasure-resilient sublinear-time graph algorithms and design an efficient algorithm for testing connectivity of graphs.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

ALLDC	Approximate Locally List Decodable Code
ALLEDC	Approximate Locally List Erasure-Decodable Code
iff . . .	if and only if
LDC . .	Locally Decodable Code
LEDC .	Locally Erasure-Decodable Code
LLDC .	Locally List Decodable Code
LLEDC	Locally List Erasure-Decodable Code
PCP . .	Probabilistically Checkable Proof
PCPP .	Probabilistically Checkable Proofs of Proximity
POT . .	Proximity Oblivious Tester
u.a.r. .	uniformly at random
w.r.t. .	with respect to
$[n]$ . . .	$\{1, 2, \dots n\}$
$\mathbb{N}$ . . .	the set $\{1, 2, \dots\}$ of natural numbers
$\mathbb{R}$ . . .	the set of real numbers
$\mathbb{F}_2$ . . .	the finite field of size 2
e	the base of the natural logarithm
log . . .	logarithm base 2
ln . . .	natural logarithm
n . . .	input size in bits, or the number of vertices in the input graph
m . . .	number of edges in the input graph
OPT . .	size of an optimal solution of an optimization problem

CHAPTER 1
Introduction

Information is now being generated and stored at an unprecedented rate. Harvesting and analyzing this information has the potential to deepen and enhance our understanding of human society and economy. It is thus critical to design efficient and scalable algorithmic solutions to solve computational problems on large datasets. Since faulty datasets are ubiquitous, the resilience of algorithms to imperfections in data is as crucial as their speed, memory efficiency, and accuracy.

Faults in datasets include both *erroneous* and *missing* entries. The problem of errors in data has been a topic of extensive research within various subareas of theoretical computer science such as robust statistics (Huber (2011)), machine learning (Kearns & Li (1993)), local error detection and correction (Yekhanin (2012); Goldreich (2011c)), and sublinear algorithms (Parnas et al. (2006)), to name a few. In contrast, the issue of missing entries in datasets has received much less attention.

Missing entries in datasets can occur for many reasons. The results of a survey can have missing responses if some of the participants feel uncomfortable answering questions that they deem sensitive. Large graphs such as social networks can have missing edges when individuals hide their friendship relations out of privacy concerns. Accidental deletion of records that are not backed up, or adversarial deletion of records by hackers can also lead to missing entries.

In this work, we propose novel computational models for algorithms to analyze massive datasets containing missing (or *erased*) entries. Further, we design efficient algorithms in our models for various well-studied problems, and investigate the relationship of our models to the existing models of algorithms for big data.

Our focus is on algorithms whose inputs can be represented as functions[1], codewords, or graphs. Moreover, a major part of the thesis is devoted to the study of sublinear-time algorithms (Goldreich (2011a,b); Ron (2009); Rubinfeld & Blais (2016); Goldreich (2017)), which are algorithms whose running time is sublinear in the length of the input representation.

First, we generalize the property testing model (Rubinfeld & Sudan (1996); Goldreich, Goldwasser, & Ron (1998)), one of the most widely studied models of sublinear-time algorithms, to account for the presence of adversarially erased function values. We design efficient erasure-resilient property testing algorithms for several fundamental properties of real-valued functions such as monotonicity, Lipschitz properties, convexity, and linearity.

We then investigate the problems of local decoding (Katz & Trevisan (2000)) and local list decoding (Sudan, Trevisan, & Vadhan (2001)) of codewords containing erasures. We show that, in some cases, these problems are strictly easier than the corresponding problems of decoding codewords containing errors. Moreover, we use this understanding to show a separation between our erasure-resilient property testing model and the (error) tolerant property testing model (Parnas, Ron, & Rubinfeld (2006)). The philosophical message of this separation is that errors occurring in large datasets are, in general harder to deal with, than erasures.

Finally, we develop models and notions to reason about algorithms that are intended to run on large graphs with missing edges. Large graphs with missing edges occur naturally in many contexts. For example, consider a social network, where a vertex corresponds to a user of the social network service and an edge corresponds to a

[1]Many objects can be viewed as functions. E.g., an n-element array of real numbers can be viewed as a function $f : [n] \to \mathbb{R}$, an image—as a map from the plane to the set of colors, and a graph—as a map from the set of vertex pairs to $\{0, 1\}$.

friendship relation between users. It is reasonable to assume that users do not always update new friendship relations on the social network service, and that sometimes they do not fully disclose their friendships because of security or privacy reasons. Hence, we can only obtain an approximation G' to the true social network G. This brings out the need for algorithms that can extract information on G by solving a problem on G'. Moreover, as the solutions output by a graph algorithm are often used in applications such as detecting communities (Newman (2004, 2006)), ranking nodes (Page et al. (1999)), and spreading influence (Kempe et al. (2003)), the solutions output by an algorithm on G' should be close to those output on G. With this motivation, we define average sensitivity, a robustness metric for graph algorithms. We discuss various useful features of our definition and design approximation algorithms with good average sensitivity bounds for several optimization problems on graphs. We also define a model of erasure-resilient property testing for graphs and design an efficient algorithm for testing connectedness.

1.1 OVERVIEW OF OUR CONTRIBUTIONS

Property testers (Rubinfeld & Sudan (1996); Goldreich et al. (1998)) form an important class of sublinear-time algorithms. We begin by modeling and studying property testing algorithms that work in the presence of partially erased data. In the *standard* property testing model, a tester accesses the input function (which formalizes a dataset) via an oracle. With very few exceptions, all property testers studied in this model rely on the oracle to provide function values at all queried domain points[2].

[2]Property testing algorithms with various distributional assumptions on the input positions that the algorithms access have been investigated, for example, in Goldreich et al. (1998); Balcan et al. (2012); Goldreich & Ron (2016). There is also a line of work, initiated by Batu et al. (2013), that studies property testers that access distributions, as opposed to fixed datasets. In this work, we focus on fixed datasets.

We define the *erasure-resilient property testing* model in which we account for *erasures* in the input function. In our model, the oracle, upon receiving a tester query, returns a special symbol $\perp$ if the point being queried is erased and the non-erased function value otherwise. We assume that erasures are made by an adversary (*adversarially*) before the tester starts querying the oracle (*obliviously*).

We design erasure-resilient property testers for a large class of properties. For some properties, our erasure-resilient testers are obtained by simply using standard testers as a black box. However, for certain more challenging properties, all existing standard testers are very likely to query specific points in the domain. If these points are erased, the algorithms break. We give efficient erasure-resilient testers for several important classes of such properties of functions, including monotonicity, the Lipschitz property, and convexity.

We extend our study of erasure-resilient property testing by relaxing the obliviousness assumption and modeling adversaries that have the power to erase input values during the execution of a testing algorithm. The motivation is that database administrators might want to selectively hide parts of a database (for privacy reasons) based on the history of queries made by an algorithm. In the *semi-oblivious erasure model*, an adversary can erase at most 1 point before seeing each query of the algorithm. We design a constant-query erasure-resilient tester against semi-oblivious adversary for linearity of Boolean functions over the Boolean hypercube. We also show that erasure-resilient sortedness testing is impossible against a semi-oblivious adversary.

Next, we study local decoding algorithms (Katz & Trevisan (2000); Yekhanin (2012); Goldreich & Levin (1989); Sudan et al. (2001)) that work in the presence of erasures. Local decoding in the presence of errors has been extensively studied, but has not been considered explicitly in the presence of erasures. We prove an analog of a famous result of Goldreich & Levin (1989) on local list decodability of the Hadamard code. Specifically, we show that the Hadamard code is locally list decodable in the presence of a constant fraction of erasures, arbitrary close to 1, with list sizes and query complexity better than in the Goldreich-Levin theorem. We also study the general relationship between local decoding in the presence of errors and in the presence of erasures. We observe that every locally (uniquely or list) decodable code that works in the presence of errors also works in the presence of twice as many erasures (with the same parameters up to constant factors). We show that there is also an implication in the other direction for locally decodable codes (with unique decoding): specifically, that the existence of a locally decodable code that works in the presence of erasures implies the existence of a locally decodable code that works in the presence of errors and has related parameters.

Enabled by our understanding of erasures and errors in local decoding, we investigate the effects of adversarial errors or erasures in inputs on the complexity of property testing algorithms. We first separate erasure-resilient testing from standard testing. Specifically, we describe a property of binary strings of length n such that it can be tested in the standard model using a constant number of queries (independent of the input length), but every erasure-resilient tester for the property has query complexity $\Omega(n)$. We then separate erasure-resilient property testing from tolerant testing by exploiting the difference in local list decoding capabilities of codes for the same fraction of erasures and errors. That is, we describe a property of binary strings of length n that can be tested with a constant number of queries in the presence of erasures, but requires $n^{\Omega(1)}$ queries for tolerant testing.

Our final focus is on developing models and metrics to systematically study algorithms for large graphs with missing edges. We study this for polynomial time graph algorithms that are intended to run on smaller subgraphs of large graphs as well as for sublinear-time algorithms whose inputs are large graphs themselves.

While running graph algorithms on subgraphs of large graphs that contain several missing edges, it is desirable to output solutions that are close to the solutions output when there are no missing edges. We formalize this idea by introducing the notion of average sensitivity of graph algorithms. It is defined as the average earth mover's distance between the output distributions of an algorithm on a graph and its subgraph obtained by removing an edge, where the average is taken over the edges removed. After deriving basic properties of average sensitivity, such as composition of algorithms with low sensitivity, we provide efficient approximation algorithms with low average sensitivity for concrete graph problems, including the minimum spanning forest problem, the global minimum cut problem, the maximum matching problem, and the minimum vertex cover problem. We also show that every algorithm for the 2-coloring problem has average sensitivity linear in the number of vertices. To show our algorithmic results, we establish and utilize the following fact: if the presence of a vertex or an edge in the solution output by an algorithm can be decided locally, then the algorithm has low average sensitivity. This fact allows us to reuse the analyses of known sublinear-time algorithms for graphs.

Our model for studying erasure-resilient sublinear-time algorithms for graphs is a direct generalization of the well-studied model of graph property testing defined by Parnas & Ron (2002). We assume that input graphs are represented by their adjacency lists. Erasures model missing entries in the adjacency lists of the input graph. Algorithms access the input graph via *degree queries* and *neighbor queries.* A degree query allows the algorithm to obtain the degree of an arbitrary vertex. A neighbor query is of the form (v, i), and the answer is the i-th entry in the adjacency list of vertex v. We show that, in our model, the problem of erasure-resiliently testing connectedness shows a threshold phenomenon. In particular, when the fraction of erasures is below a certain threshold, connectedness can be tested using a constant number of queries independent of the size of the input graph. However, when the fraction of erasures is at least that threshold, testing connectedness can require examining the entire graph.

1.2 ERASURE-RESILIENT PROPERTY TESTING

Erasure-resilient property testing model generalizes the standard property testing model of Rubinfeld & Sudan (1996) and Goldreich, Goldwasser, & Ron (1998) to account for the presence of oblivious adversarial erasures in the input.

Given a parameter $\alpha \in [0, 1)$, we say that a function is *α-erased* if at most an α fraction of its domain points are marked as "erased" or protected (that is, an algorithm is denied access to these values). An algorithm that gets oracle access to an α-erased function as its input does not know which values are erased until it queries the corresponding domain points. For each queried point x, the algorithm either learns $f(x)$ or, if x is an erased point, gets back a special symbol $\perp$.

An α-erasure-resilient ε-tester for a property $\mathcal{P}$ is given parameters $\alpha \in [0, 1), \varepsilon \in (0, 1)$ satisfying $\alpha + \varepsilon < 1$, along with oracle access to an α-erased function f. The tester has to accept with high probability if f can be completed to a function on the whole domain that satisfies the desired property $\mathcal{P}$ and reject with high probability if every completion of f has to be modified in at least an ε fraction of the domain in order to satisfy $\mathcal{P}$. In the latter case, we say that f is ε-far from $\mathcal{P}$. As our focus is on property testers that work in the presence of adversarial erasures, the query complexity of an α-erasure-resilient tester is the number of queries it makes in the *worst case* over all α-erased input functions.

Generic Transformations. Our first goal while designing erasure-resilient testers is to understand which existing algorithms in the standard property testing model can be easily made erasure-resilient. We show how to obtain erasure-resilient testers for some properties by using standard testers for these properties as black box. Our transformations apply to *sample-based testers*, which are testers that query uniformly and independently sampled points[3]. More specifically, our first transformation works for *proximity oblivious testers* (POTs), defined by Goldreich & Ron (2011), that are, in addition, restricted to be sample-based. Our second transformation applies to sample-based testers for a class of properties that we call *extendable*. Loosely speaking, a property is extendable if (1) a function satisfying the property on a subdomain can be extended to a function satisfying the same property on the whole domain, and (2) a function that is ε-far from the property on a subdomain cannot be extended to a function on the whole domain that satisfies the property without changing the values on at least an ε fraction of positions on the subdomain. Extendable properties are a generalization of a class of properties defined by Jha & Raskhodnikova (2013). Using our second transformation, we are able to obtain erasure-resilient testers for properties such as being a low-degree univariate polynomial (Rubinfeld & Sudan (1996)), monotonicity over general poset domains (Fischer et al. (2002)), convexity of black and white images (Berman et al. (2016b)), and Boolean functions over $[n]$ with at most k runs of 0s and 1s.

[3]Sample-based testers were first considered by Goldreich et al. (1998). A systematic study of sample-based testers was initiated by Goldreich & Ron (2016) and continued by Fischer et al. (2015). Sample-based testers are also called uniform testers by Berman et al. (2016a,b,c) in their papers on testing image properties.

Erasure-Resilient Testers for more Challenging Properties. One challenge in designing erasure-resilient testers by using existing algorithms in the standard model as a starting point is that many existing algorithms are more likely to query certain points in the domain. Therefore, if these points are erased, the algorithms break. Specifically, some of the previously known optimal algorithms for testing whether a list of numbers is sorted[4] have this feature. Moreover, it is known that an algorithm that makes uniformly random queries to test sortedness is far from optimal: it needs $\Theta(\sqrt{n})$ queries instead of the optimal $\Theta(\log n)$ for n-element lists (Ergün et al. (2000); Fischer (2004)).

There are a number of well studied properties for which all known optimal algorithms heavily rely on querying specific points. Most prominent examples include monotonicity, the Lipschitz properties and, more generally, bounded-derivative properties of real-valued functions on $[n]$ and $[n]^d$, as well as convexity of real-valued functions on $[n]$. It is also especially challenging to deal with real-valued functions in our model, because there are many possibilities for erased values. We give efficient erasure-resilient testers for all aforementioned properties of real-valued functions.

1.3 ERASURE-RESILIENT PROPERTY TESTING AGAINST SEMI-OBLIVIOUS ADVERSARY

In the erasure-resilient property testing model discussed above, we assumed that all the erasures are made obliviously, that is, before the algorithm starts querying the oracle. We study a generalization of this model by considering a *semi-oblivious adversary*, that, before seeing each query of the tester, erases at most 1 point.

We study *erasure-resilient testing against semi-oblivious adversary* for sortedness

[4]There are at least four different algorithms (Ergün et al. (2000); Bhattacharyya et al. (2012); Chakrabarty & Seshadhri (2013); Belovs (2018)) for this problem.

of real-valued arrays (Ergün et al. (2000)), and linearity of Boolean functions over the Boolean hypercube (Blum et al. (1993)). We show that it is *impossible* to test sortedness against a semi-oblivious adversary. However, we show that linearity can be erasure-resiliently tested with a constant number of queries against semi-oblivious adversary.

1.4 ERASURES VERSUS ERRORS IN LOCAL DECODING

Intuitively, a family of codes is *locally decodable* in the presence of a specified type of corruptions (erasures or errors) if there exists an algorithm that, given oracle access to a codeword with a limited fraction of specified corruptions, can decode each desired character of the encoded message with high probability after querying a small number of characters in the corrupted codeword. In other words, we can simulate oracle access to the message by using oracle access to a corrupted codeword. This notion can be extended to local *list* decoding by requiring the algorithm to output a list of descriptions of local decoders.

Intuitively, a family of codes is *locally list decodable* in the presence of a specified type of corruptions if there exists an algorithm that, given oracle access to a corrupted codeword w, outputs a list of algorithms such that for each message x whose encoding sufficiently agrees with w, there is an algorithm in the list that, given oracle access to w, can simulate oracle access to x. In addition to the usual quantities studied in the literature on error-correcting codes (such as the fraction of corruptions a code can handle, its rate and efficiency of decoding), the important parameters in local list decoding are the number of queries that the algorithms make to w and the list size.

An approximate locally list decodable code is defined identically to a locally list decodable code except that the algorithms output by a decoder for such a code simulate oracle access to strings that are close in Hamming distance to the original messages.

Local Erasure Decoding versus Local Decoding. First, we study the general relationship between local decoding in the presence of errors and in the presence of erasures. We observe that every locally list decodable code that works in the presence of errors also works in the presence of twice as many erasures (with the same parameters up to constant factors). Similar observations hold for local unique decoding and approximate local list decoding. We ask if locally list decodable codes or approximate locally list decodable codes that work in the presence of erasures can have significantly smaller list sizes and query complexity than locally list decodable codes or approximate locally list decodable codes of the same rate that work in the presence of errors. We also prove that such a statement cannot hold for the case of local unique decoding: specifically, we show that if a code is locally unique erasure-decodable, then there exists another comparable code that is locally unique decodable (up to minor losses in parameters).

Local List Erasure-Decodability of Hadamard Code. Local list decodability of the Hadamard code (see Definition 5.1.9) in the presence of errors is a famous result of Goldreich & Levin (1989). However, (local list) decoding of the Hadamard code is impossible when the fraction of errors reaches or exceeds $1/2$. In contrast, we show that the Hadamard code is locally list decodable in the presence of any constant fraction of erasures in $[0, 1)$. Moreover, the list size and the query complexity for our decoder is better than for the Goldreich-Levin decoder: for our decoder, both quantities are inversely proportional to the fraction of input that has not been corrupted, whereas for the Goldreich-Levin decoder they are quadratically larger. Thus, our Hadamard decoder demonstrates that a square-root reduction in the list size and query complexity in local list decoding can be achieved for some settings of parameters when we move from errors to erasures.

1.5 SEPARATING ERASURE-RESILIENT TESTING FROM STANDARD AND (ERROR) TOLERANT TESTING.

Investigating the effects of adversarial input corruption on the complexity of sublinear algorithms is a problem of fundamental importance. It is this motivation that spurred the generalization of property testing, one of the most widely studied models of sublinear-time algorithms, to (error) tolerant testing (Parnas et al. (2006)) and erasure-resilient testing.

We separate erasure-resilient testing both from standard testing as well as tolerant testing. Specifically, we show two properties of binary strings of length n such that: for the first property, standard testing has constant query complexity but erasure-resilient testing requires $\tilde{\Omega}(n)$ queries; for the second property, erasure-resilient testing has constant query complexity but tolerant testing requires $n^{\Omega(1)}$ queries. Our results place erasure-resilient property testing model in a larger picture of two widely studied property testing models. At the heart of our separation between erasure-resilient testing and tolerant testing is our understanding of the complexity of local decoding decoding from erasures as opposed to errors.

Standard Testing and Erasure-Resilient Testing. Fischer & Fortnow (2006) separated standard and tolerant testing by describing a property over binary strings of length n that can be tested using a constant number of queries in the standard model but requires $n^{\Omega(1)}$ queries to test tolerantly. The construction of our property that separates standard and erasure-resilient testing models is identical in spirit to that of Fischer & Fortnow (2006). However, our result is stronger than the result of Fischer & Fortnow (2006), as erasure-resilient testing is at least as easy as tolerant testing.

The property $\mathcal{P}$ that we use to separate erasure-resilient testing and standard testing is constructed from another property $\mathcal{R}$ such that the query complexity of testing $\mathcal{R}$ in the standard model is linear in the input size. However, $\mathcal{R}$ has the feature that, given oracle access to an additional *proof* string, it takes only a constant number of queries to test $\mathcal{R}$, where the query complexity includes the number of queries made to the proof. The property $\mathcal{P}$ is defined as the set of all strings where the first part consists of repetitions of a string that satisfies $\mathcal{R}$, and the second part is the corresponding proof string. It follows that $\mathcal{P}$ is easy to test in the standard model. However, if the *sensitive region* containing the proof is erased, erasure-resilient testing of $\mathcal{P}$ reduces to testing of $\mathcal{R}$ without access to a proof, which requires a high query complexity.

Erasure-Resilient Testing and Tolerant Testing. The starting point in our construction of the property $\mathcal{P}'$ to separate erasure-resilient and tolerant testing is the property $\mathcal{P}$ that we used to separate erasure-resilient testing from standard testing. The additional idea here is to encode sensitive region of strings (without which testing becomes hard) with an error correcting code, specifically a locally list decodable code with certain desirable properties.

For our construction, we need a code that exhibits a difference in its local list decoding capabilities for the same fraction of erasures and errors. Specifically, we want, for some constant α, q and L, a code that can be decoded from an α fraction of erasures with q queries and lists of size L, but cannot be decoded from an α fraction of errors. As the Hadamard code satisfies these requirements, we define our property

$\mathcal{P}'$ by encoding the sensitive region of strings with the Hadamard code. We show that $\mathcal{P}'$ can be tested in the erasure-resilient model with a constant number of queries; but every tolerant tester for $\mathcal{P}'$ has query complexity polylogarithmic in the length of the input.

Next, we strengthen the separation to obtain a property $\mathcal{P}''$ that can be tested using a constant number of queries in the presence of erasures, but requires as many queries as possible to test tolerantly. In our construction, the lower bound on the number of queries needed for tolerant testing is determined by the rate of the code used. Since the Hadamard code has low rate, we only get a polylogarithmic lower bound on the query complexity of tolerant testing. To obtain a lower bound of $n^{\Omega(1)}$, we would need a code of polynomial rate. The question of whether there is a locally list erasure-decodable code with constant α, q, and L, and has polynomial rate remains open. In fact, a locally list decodable code with such parameters is the holy grail of research on local decoding.

We circumvent the above difficulty by starting out with a property of binary strings that has an efficient tester whose queries to a sensitive region of the input are *nearly uniformly* distributed. This implies that testing remains easy even if a constant fraction of the sensitive region is corrupted. We construct our separating property $\mathcal{P}''$ by encoding the sensitive region using an approximate locally list decodable with constant α, q, and L, and polynomial rate. We show that $\mathcal{P}''$ can be erasure-resiliently tested using a constant number of queries but needs $n^{\Omega(1)}$ queries in order to be tested tolerantly, thus obtaining the desired strengthened separation.

1.6 AVERAGE SENSITIVITY OF GRAPH ALGORITHMS

We consider graph algorithms whose outputs are typically vertex sets or edges sets and assume that the outputs are represented appropriately using binary strings. We assume that the n-node input graph G' at hand is a randomly chosen (large) subgraph of an unknown true graph G.

Intuitively, a deterministic algorithm $\mathcal{A}$ is stable-on-average when the Hamming distance $d_{\mathrm{Ham}}\big(\mathcal{A}(G), \mathcal{A}(G')\big)$ is small, where $\mathcal{A}(G)$ and $\mathcal{A}(G')$ are outputs of $\mathcal{A}$ on G and G', respectively. More specifically, for an integer $k \geq 1$, we say that the *k-average sensitivity* of a deterministic algorithm $\mathcal{A}$ is

$$\mathbb{E}_{\{e_1,\ldots,e_k\}\sim\binom{E}{k}}\big[d_{\mathrm{Ham}}\big(\mathcal{A}(G), \mathcal{A}(G - \{e_1, \ldots, e_k\})\big)\big] \tag{1.1}$$

for every graph $G = (V, E)$, where $G - F$ for an edge set F is the subgraph obtained from G by removing F, and $e_1, \ldots, e_k$ are sampled from E uniformly at random without replacement. When $k = 1$, we call the k-average sensitivity simply the *average sensitivity.* We say that algorithms with low k-average sensitivity are *k-stable-on-average.* We also define average sensitivity for randomized algorithms by replacing the Hamming distance above with the earth mover's distance between output distributions.

Features of Average Sensitivity. We show that it is possible to obtain meaningful bounds on k-average-sensitivity of algorithms using corresponding bounds on their average sensitivity. We also show that stable-on-average algorithms can be sequentially composed to obtain algorithms that are also stable-on-average, albeit with a worse average sensitivity. Further, we can bound the average sensitivity of an algorithm sampled according to a distribution over multiple stable-on-average algorithms, where the distribution may be determined by parameters of the input graph. We call this the parallel composition theorem and use it extensively in constructing stable-on-average algorithms. We use all of the above properties to design stable-on-average algorithms from simpler algorithms that are stable-on-average.

Connection to Local Algorithms. In the model of local computation algorithms (Nguyen & Onak (2008); Rubinfeld et al. (2011); Yoshida et al. (2012)), the aim is to answer queries about elements in the solution of an algorithm by making as few queries to the input as possible. Roughly speaking, we show that algorithms that output solutions with *strong locality* properties have low average sensitivity. Specifically, if there is a local algorithm $\mathcal{O}$ that gets access to an input graph G and answers queries about the solution output by an algorithm $\mathcal{A}$ on G, then the average sensitivity of $\mathcal{A}$ is at most the worst-case number of queries that $\mathcal{O}$ makes to G to answer a single query about the solution.

Stable-on-average Algorithms for Concrete Problems. We design stable-on-average algorithms for several graph problems such as minimum spanning forest, global minimum cut, maximum matching, and minimum vertex cover. For the minimum spanning forest problem, we show that Kruskal's algorithm has average sensitivity $o(1)$. We design approximation algorithms with average sensitivity $o(\mathsf{OPT})$ for all the other aforementioned problems. In comparison, all known polynomial time algorithms for these problems have average sensitivity $\Omega(n)$. Our results exhibit a trade-off between the average sensitivity and approximation guarantee.

1.7 ERASURE-RESILIENT GRAPH PROPERTY TESTING

We define the erasure-resilient model of graph property testing with the aim of understanding global properties of a large graph with missing edges by observing only a small portion of the whole graph. Our model is a generalization of the (general) model of graph property testing (Parnas & Ron (2002)).

In our model, graphs are represented as adjacency lists. Let $\alpha \in [0, 1)$ be a parameter. A graph G is α-erased if at most an α fraction of the entries in its adjacency lists are erased. An algorithm A that has oracle access to an α-erased graph G over a vertex set $[n]$ can make *degree queries* and *neighbor queries*. A degree query is of the form $v \in [n]$, and the oracle returns the degree of v. A neighbor query is of the form (v, i), and the answer is the i^{th} entry in the adjacency list of vertex v. A completion of an α-erased graph G is an assignment of vertex labels to erased entries in the adjacency lists of G such that the adjacency lists obtained after the completion corresponds to a graph G'. An α-erased graph G satisfies a property $\mathcal{P}$ if there is a completion of G that satisfies $\mathcal{P}$. It is ε-far from $\mathcal{P}$ if every completion has to be modified in at least an ε fraction of edges to satisfy $\mathcal{P}$.

We study the problem of testing connectivity in our model. We design an α-erasure-resilient ε-tester for connectivity with query complexity $O\left(\frac{1}{((\varepsilon-\alpha)\overline{d})^3}\right)$ that works for all $\alpha < \varepsilon$, where $\overline{d}$ is the average degree of the graph. We also show that α-erasure-resiliently ε-testing connectivity requires $\Omega(m)$ queries whenever $\alpha \geq \varepsilon$, where m is the number of edges in every completion of the graph.

1.8 ORGANIZATION AND BIBLIOGRAPHIC NOTES

Chapter 2 sets up some of the notation and definitions used in the thesis. We define and study the erasure-resilient property testing model in Chapter 3. The contents of this chapter are based on joint work with Kashyap Dixit, Sofya Raskhodnikova, and Abhradeep Thakurta (Dixit et al. (2018)). Chapter 4 contains our results on erasure-resilient property testing against semi-oblivious adversary and is joint (unpublished) work with Iden Kalemaj and Sofya Raskhodnikova. The idea of studying erasure-resilient property testing under semi-oblivious adversarial erasures was suggested to us by Kobi Nissim. Chapter 5 on the role of erasures and errors in local decoding is joint work with Sofya Raskhodnikova and Noga Ron-Zewi (Raskhodnikova et al. (2019)). Observation 5.1.5 therein is based on an idea suggested by Venkatesan Guruswami. Chapter 6 contains our results separating erasure-resilient property testing from the standard and (error) tolerant models of property testing, and is based on both Dixit et al. (2018) and Raskhodnikova et al. (2019). In particular, the result separating erasure-resilient testing from standard testing in Section 6.2 is an improvement of a similar result in Dixit et al. (2018), whereas the separation results in Section 6.3 come from Raskhodnikova et al. (2019). Chapter 7 is joint (unpublished) work with Yuichi Yoshida (Varma & Yoshida (2019)). It consists of our definition of average sensitivity of graph algorithms and our stable-on-average algorithms for several optimization problems on graphs. Chapter 8 contains our investigation of erasure-resilient sublinear-time algorithms for graphs and is based on joint (unpublished) work with Amit Levi, Ramesh Krishnan S. Pallavoor, and Sofya Raskhodnikova (Levi et al. (2019)). Chapter 9 is devoted to conclusions and open problems.

CHAPTER 2
Preliminaries

Property Testing and Tolerant Property Testing. A property $\mathcal{P}$ is a set. As mentioned earlier, we restrict our attention to properties of functions. Let $\mathcal{P}$ be a property of functions from $\mathcal{D}$ to $\mathcal{R}$. A function $f : \mathcal{D} \to \mathcal{R}$ satisfies $\mathcal{P}$ if $f \in \mathcal{P}$. The distance of f from $\mathcal{P}$, denoted $\text{dist}(f, \mathcal{P})$, is defined as the minimum number of points in the domain of f whose values need to be changed in order to satisfy $\mathcal{P}$. Let $\varepsilon \in (0, 1)$. A function f is ε-far from $\mathcal{P}$ if $\text{dist}(f, \mathcal{P}) \geq \varepsilon|\mathcal{D}|$. It is ε-close otherwise.

Definition 2.0.1 (Property Tester). *Let $\varepsilon \in (0, 1)$. An ε-tester for the property $\mathcal{P}$ gets the parameter ε and oracle access to a function $f : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$. It outputs, with probability*[1] *at least $2/3$,*

- ***accept** if f satisfies $\mathcal{P}$;*
- ***reject** if f is ε-far from $\mathcal{P}$.*

The tester has 1-*sided error* if the first item holds with probability 1.

Definition 2.0.2 (Tolerant Property Tester). *Let $\varepsilon_1, \varepsilon_2 \in (0, 1)$ such that $\varepsilon_1 < \varepsilon_2$. An $(\varepsilon_1, \varepsilon_2)$-tolerant tester for the property $\mathcal{P}$ gets parameters $\varepsilon_1, \varepsilon_2$ and oracle access to a function $f : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$. It outputs, with probability at least $2/3$,*

- ***accept** if f is ε_1-close to $\mathcal{P}$;*
- ***reject** if f is ε_2-far from $\mathcal{P}$.*

A tolerant tester is fully tolerant if it works for all $\varepsilon_1, \varepsilon_2 \in (0, 1)$ such that $\varepsilon_1 < \varepsilon_2$.

[1] In general, the error probability can be any $\delta \in (0, 1)$. For simplicity, we formulate our model and the results with $\delta = 1/3$. To get results for general δ, by standard arguments, it is enough to multiply the complexity of an algorithm by $\log 1/\delta$.

Definition 2.0.3 (Nonadaptive and Adaptive Testers). *A tester (or tolerant tester) is nonadaptive if the queries made by the tester (or tolerant tester) do not depend on the answers to the previous queries, and adaptive otherwise.*

Total Variation Distance. The total variation distance between two probability measures P_1 and P_2 on a sigma-algebra $\mathcal{F} \subseteq 2^{\Omega}$, denoted as $d_{\mathrm{TV}}(P_1, P_2)$, is defined as

$$\sup_{A \in \mathcal{F}} |P_1(A) - P_2(A)|.$$

When P_1 and P_2 are distributions over a discrete sample space Ω, which is the case in this thesis for the most part, $d_{\mathrm{TV}}(P_1, P_2)$ is equal to

$$\frac{1}{2} \sum_{\omega \in \Omega} |P_1(\omega) - P_2(\omega)|.$$

Earth Mover's Distance. The earth mover' distance is also known as the Wasserstein metric. Consider two probability distributions P_1 and P_2 P_1 and P_2 defined on two sigma-algebras $\mathcal{F}_1 \subseteq 2^{\Omega_1}$ and $\mathcal{F}_2 \subseteq 2^{\Omega_2}$, respectively. Let $d : \Omega_1 \times \Omega_2 \to \mathbb{R}$ be a distance. The earth mover's distance between P_1 and P_2 is the optimal cost of converting P_1 to P_2 (or vice versa), where the cost of moving a probability in the amount of p from $\omega_1 \in \Omega_1$ to $\omega_2 \in \Omega_2$ is equal to $p \cdot d(\omega_1, \omega_2)$.

CHAPTER 3
Erasure-Resilient Property Testing

In this chapter, we define and study a model of function property testing algorithms that are resilient to the presence of *adversarial erasures* in the input.

3.1 ERASURE-RESILIENT TESTING MODEL

We formalize our erasure-resilient model for function property testing.

Definition 3.1.1 (α-erased function)**.** *Let $\mathcal{D}$ be a domain, $\mathcal{R}$ be a range, and $\alpha \in [0, 1)$. A function $f : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$ is α-erased if f evaluates to $\perp$ on at most an α fraction of domain points. The points on which f evaluates to $\perp$ are called erased. The set of remaining (nonerased) points is denoted by $\mathcal{N}$.*

Note that an α-erased function has *at most* an α fraction of its values erased. In particular, a function with no erasures is also an α-erased function for all $\alpha \in [0, 1)$.

A function $f' : \mathcal{D} \to \mathcal{R}$ that differs from a function f only on points erased in f is called a *completion* of f. The (Hamming) distance of an α-erased function $f : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$ from a property (set) $\mathcal{P}$ is the minimum number of points on which every completion of f needs to be changed to satisfy $\mathcal{P}$. The relative Hamming distance of f from $\mathcal{P}$ is the aforementioned quantity normalized by the size of the domain $\mathcal{D}$.

An α-erased function $f : \mathcal{D} \to \mathcal{R}$ is ε-far from a property $\mathcal{P}$ if the relative Hamming distance of f from $\mathcal{P}$ is at least ε. Since one can always assume that f is "correct" on the erased points, every completion of f has to be changed in at least $\varepsilon|\mathcal{D}|$ points in order to satisfy $\mathcal{P}$. This imposes the natural restriction of $\alpha + \varepsilon < 1$ on α and ε.

Definition 3.1.2 (Erasure-Resilient Tester). *An α-erasure-resilient ε-tester of property $\mathcal{P}$ gets input parameters $\alpha \in [0,1)$, $\varepsilon \in (0,1)$ satisfying $\alpha + \varepsilon < 1$, and oracle access to an α-erased function $f : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$. It outputs, with probability at least 2/3,*

- ***accept*** *if there is a completion $f' : \mathcal{D} \to \mathcal{R}$ of f that satisfies $\mathcal{P}$;*
- ***reject*** *if every completion $f' : \mathcal{D} \to \mathcal{R}$ of f needs to be changed on at least εn points to satisfy $\mathcal{P}$ (that is, f' is ε-far from $\mathcal{P}$).*

The tester has 1*-sided error if the first item holds with probability 1. The tester is nonadaptive if the queries made by the tester do not depend on the answers to the previous queries, and adaptive otherwise.*

Remark 3.1.3. *Let $\alpha \in [0,1)$, $\varepsilon \in (0,1)$ satisfying $\alpha + \varepsilon < 1$. It is natural to assume that an α-erasure-resilient ε-tester for a property $\mathcal{P}$ is also an α'-erasure-resilient ε'-tester for all $\alpha' \in [0, \alpha]$, $\varepsilon' \in [\varepsilon, 1)$ satisfying $\alpha' + \varepsilon' < 1$.*

3.2 PROPERTIES THAT WE STUDY

Next we define properties of real-valued functions considered in this chapter and summarize previous work on testing them. Most properties of real-valued functions studied in the property testing framework are for functions over the *line* domain $[n]$ and, more generally, the *hypergrid* domain $[n]^d$.

Definition 3.2.1 (Hypergrid, Line). *Given $n, d \in \mathbb{N}$, the hypergrid of size n and dimension d is the set $[n]^d$ associated with an order relation $\preceq$, such that $x \preceq y$ for all $x, y \in [n]^d$ iff $x_i \leq y_i$ for all $i \in [d]$, where x_i (respectively y_i) denotes the i^{th} coordinate of x (respectively, y). The special cases $[n]$ and $[2]^d$ are called the line and hypercube, respectively.*

We also consider domains that are subsets of $[n]^d$ to be able to handle arbitrary erasures on $[n]^d$.

Monotonicity. Monotonicity of functions, first studied in the context of property testing by Goldreich et al. (2000b), is one of the most widely investigated properties in this model. Ergün et al. (2000); Dodis et al. (1999); Lehman & Ron (2001); Fischer et al. (2002); Ailon & Chazelle (2006); Fischer (2004); Halevy & Kushilevitz (2008); Batu et al. (2005); Parnas et al. (2006); Ailon et al. (2007); Bhattacharyya et al. (2012); Briët et al. (2012); Blais et al. (2012); Chakrabarty & Seshadhri (2013, 2014); Blais et al. (2014); Chen et al. (2014); Belovs & Blais (2015); Chakrabarty et al. (2017); Chen et al. (2015); Belovs & Blais (2016); Chakrabarty & Seshadhri (2016); Khot et al. (2018); Black et al. (2018); Pallavoor et al. (2018); Belovs (2018); Chakrabarty & Seshadhri (2019) is a (by no means exhaustive) list of works that study monotonicity testing.

A function $f : \mathcal{D} \to \mathbb{R}$, defined on a partially ordered domain $\mathcal{D}$ with order $\preceq$, is monotone if $x \preceq y$ implies $f(x) \leq f(y)$ for all $x, y \in \mathcal{D}$. The query complexity of testing monotonicity of functions $f : [n] \to \mathbb{R}$ is $\Theta(\log n/\varepsilon)$ Ergün et al. (2000); Fischer (2004); for functions $f : [n]^d \to \mathbb{R}$, it is $\Theta(d \log n/\varepsilon)$ as shown by Chakrabarty & Seshadhri (2013, 2014). For functions over arbitrary partially ordered domains $\mathcal{D}$, it is $O(\sqrt{|\mathcal{D}|/\varepsilon})$ and was proven by Fischer et al. (2002).

Lipschitz Properties. Lipschitz continuity is defined for functions between arbitrary metric spaces, but was specifically studied for real-valued functions on hypergrid domains by Jha & Raskhodnikova (2013); Awasthi et al. (2016); Chakrabarty & Seshadhri (2013); Dixit et al. (2013); Blais et al. (2014); Chakrabarty et al. (2017) motivated by its applications to privacy, which was made explicit by Jha & Raskhodnikova (2013); Dixit et al. (2013). For $\mathcal{D} \subseteq [n]^d$ and $c \in \mathbb{R}$, a function $f : \mathcal{D} \to \mathbb{R}$ is c-Lipschitz if $|f(x) - f(y)| \leq c \cdot ||x - y||_1$ for all $x, y \in \mathcal{D}$, where $||x - y||_1$ is the L_1 distance between x and y. More generally, f is (α, β)-Lipschitz, where $\alpha < \beta$, if $\alpha \cdot ||x - y||_1 \leq |f(x) - f(y)| \leq \beta \cdot ||x - y||_1$ for all $x, y \in [n]^d$. Chakrabarty & Seshadhri (2013) proved that all (α, β)-Lipschitz properties can be tested with $O(d \log n/\varepsilon)$ queries.

Bounded Derivative Properties (BDPs). The class of BDPs, defined by Chakrabarty et al. (2017), is a natural generalization of monotonicity and the (α, β)-Lipschitz properties. An ordered set $\mathbf{B}$ of $2d$ functions $l_1, u_1, l_2, u_2, \ldots, l_d, u_d : [n-1] \to \mathbb{R} \cup \{\pm\infty\}$ is a *bounding family* if for all $r \in [d]$ and $y \in [n-1]$, $l_r(y) < u_r(y)$. Let $\mathbf{B}$ be a bounding family of functions and let $\mathbf{e}_r$ be the unit vector along dimension r. The property $\mathcal{P}(\mathbf{B})$ of being $\mathbf{B}$-*derivative bounded* is the set of functions $f : [n]^d \to \mathbb{R}$ such that $l_r(x_r) \leq f(x + \mathbf{e}_r) - f(x) \leq u_r(x_r)$ for all $r \in [d]$ and $x \in [n]^d$ with $x_r \neq n$, where x_r is the r^{th} coordinate of x.

Consider a graph $\mathcal{H}$ with vertex set $[n]^d$ and edges in both directions between every pair of points in $[n]^d$ that differ in exactly one coordinate. Then the value $u_r(x_r)$ is the upper bound on the increase in function value along the edge $(x, x + e_r)$ and $-l_r(x_r)$ is the upper bound on the increase in function value along the edge $(x + e_r, x)$. A bounding family $\mathbf{B} = \{l_1, u_1, \ldots, l_d, u_d\}$ defines a quasi-metric

$$\mathfrak{m}_{\mathbf{B}}(x, y) := \sum_{r : x_r > y_r} \sum_{t = y_r}^{x_r - 1} u_r(t) - \sum_{r : x_r < y_r} \sum_{t = x_r}^{y_r - 1} l_r(t)$$

over points $x, y \in [n]^d$. Chakrabarty et al. (2017) observe that for $\mathcal{D} = [n]^d$, a function $f : \mathcal{D} \to \mathbb{R}$ satisfies $\mathcal{P}(\mathbf{B})$, the bounded derivative property defined by $\mathbf{B}$, iff $\forall x, y \in \mathcal{D},\ f(x) - f(y) \leq \mathfrak{m}_{\mathbf{B}}(x, y)$. To get an intuition about this observation, note that $\mathfrak{m}_{\mathbf{B}}(x,y)$ is the upper bound dictated by the functions in $\mathbf{B}$ on the amount by which the value of f can increase along a shortest path from y to x in the graph $\mathcal{H}$. We use this characterization as our definition of BDPs for functions over arbitrary $\mathcal{D} \subseteq [n]^d$.

The bounding family for monotonicity is obtained by setting $l_r(y) = 0$ and $u_r(y) = \infty$ for all $r \in [d]$, and for the (α, β)-Lipschitz property, by setting $l_r(y) = \alpha$ and $u_r(y) = \beta$ for all $r \in [d]$. In general, different bounding families allow a function to be monotone in one dimension, (α, β)-Lipschitz in another dimension and so on. Chakrabarty et al. (2017) showed that for every BDP $\mathcal{P}$, the complexity of testing $\mathcal{P}$ for functions $f : [n]^d \to \mathbb{R}$ is $\Theta(d \log n/\varepsilon)$.

Convexity of Functions. A function $f : \mathcal{D} \to \mathbb{R}$ is convex if $f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{D}$ and $t \in [0,1]$. If $\mathcal{D} \subseteq [n]$, equivalently, f is convex if $\frac{f(y)-f(x)}{y-x} \leq \frac{f(z)-f(y)}{z-y}$ for all $x < y < z$. Parnas, Ron, & Rubinfeld (2003) gave a convexity tester for functions $f : [n] \to \mathbb{R}$ with query complexity $O(\log n/\varepsilon)$. Blais, Raskhodnikova, & Yaroslavtsev (2014) gave an $\Omega(\log n)$ bound for nonadaptive testers for this problem.

3.3 OUR RESULTS

We give efficient erasure-resilient testers for all properties discussed in Section 3.2.

Monotonicity on the Line. We start by giving (in Section 3.5) an erasure-resilient monotonicity tester on $[n]$.

Theorem 3.3.1 (Monotonicity tester on the line)**.** *There exists a* 1*-sided error* α*-erasure-resilient* ε*-tester for monotonicity of real-valued functions on the line* $[n]$ *that works for all* $\alpha \in [0,1), \varepsilon \in (0,1)$ *such that* $\alpha + \varepsilon < 1$ *with query complexity* $O\left(\frac{\log n}{\varepsilon}\right)$.

Without erasure resilience, the complexity of testing monotonicity of functions $f : [n] \to \mathbb{R}$ is $\Theta(\log n/\varepsilon)$, which follows from the works of Ergün et al. (2000); Fischer (2004). Thus, the query complexity of our erasure-resilient tester has optimal dependence on the domain size and on ε.

The starting point of our algorithm is the sortedness tester of Ergün et al. (2000). This tester picks a random element of the input array and performs a binary search for that element. It rejects if the binary search does not lead to the right position. The first challenge is that the tester always queries the middle element of the array and is very likely to query other elements that are close to the root in the binary search tree. So, it will break if these elements are erased. To make the tester resilient to erasures, we randomize the binary tree with respect to which it performs the binary search. The second challenge is that the tester does not know which points are erased. To counteract that, our tester samples points from appropriate intervals until it encounters a nonerased point.

Bounding the expected query complexity of our tester (Claim 3.5.2) is the most interesting part of the analysis. We view the tester as performing a binary search for a uniformly random nonerased point in the array (obtained via sampling), where at every step of the binary search, the nonerased point that *guides* the search at that step is sampled uniformly at random from the interval at that step. The intuition behind our analysis is that such a randomized binary search for a uniformly random search point is biased towards visiting intervals containing a larger fraction of nonerased points. In other words, conditioned on picking a specific nonerased point to split the current interval, the probability of the search point being in the left (or right) subinterval of the current interval is proportional to the fraction of nonerased points in that subinterval.

BDPs on the Hypergrid. In Sections 3.6-3.7, we generalize our monotonicity tester in two ways: (1) to work over general hypergrid domains, and (2) to apply to all BDPs. We achieve it by giving (1) a reduction from testing BDPs on the line to testing monotonicity on the line that applies to erasure-resilient testers and (2) an erasure-resilient version of the dimension reduction by Chakrabarty et al. (2017).

Theorem 3.3.2 (BDP tester on the hypergrid)**.** *For every BDP $\mathcal{P}$ of real-valued functions on the hypergrid $[n]^d$, there exists a 1-sided error α-erasure-resilient ε-tester that works for all $\alpha \in [0,1), \varepsilon \in (0,1)$, where $\alpha \leq \varepsilon/968d$ and $\varepsilon + \alpha < 1$, with query complexity $O\left(\frac{d \log n}{\varepsilon}\right)$.*

Every known tester of a BDP for real-valued functions over hypergrid domains work by sampling an *axis-parallel line* uniformly at random and checking for violations to the property on the sampled line. Our erasure-resilient BDP testers also follow this paradigm. To check for violations to the specified property on the sampled line, we use one iteration of our BDP tester for the line. We show (in Section 3.7.4) the existence of α-erased functions $f : \{0,1\}^d \to \mathbb{R}$ that are ε-far from monotone for $\alpha = \Theta(\varepsilon/\sqrt{d})$ but do not have even a single violation to monotonicity along any of the axis parallel lines (which are the edges of the hypercube, in this case). It implies that every tester for monotonicity that follows the above paradigm fails when $\alpha = \Omega(\varepsilon/\sqrt{d})$. Thus, some restriction on α in terms of d and ε is necessary for such testers.

Convexity on the Line. Finally, in Section 3.8, we develop additional techniques to design a tester for convexity (which is not a BDP) on the line. The asymptotic query complexity of our tester is identical to that of the standard convexity tester of Parnas et al. (2003). Blais et al. (2014) proved that the dependence on n is optimal for nonadaptive testers of convexity over the line. Moreover, the tester of Parnas et al. (2003) is conjectured to be optimal in the standard model.

Theorem 3.3.3 (Convexity tester on the line)**.** *There exists a* 1*-sided error* α*-erasure-resilient* ε*-tester for convexity of real-valued functions on the line* $[n]$ *that works for all* $\alpha \in [0,1), \varepsilon \in (0,1)$ *such that* $\alpha + \varepsilon < 1$ *with query complexity* $O\left(\frac{\log n}{\varepsilon}\right)$.

Our algorithm for testing convexity combines ideas on testing convexity from Parnas et al. (2003), testing sortedness from Ergün et al. (2000), and our idea of randomizing the search. The tester of Parnas et al. (2003) traverses a uniformly random path in a binary tree on the array $[n]$ by selecting one of the half-intervals of an interval uniformly at random at each step. Instead of doing this, our tester samples a uniformly random nonerased search point and traverses the path to that point in a random binary search tree just as in our modification of the tester of Ergün et al. (2000). This is done to bias our algorithm to traverse paths containing intervals that have a larger fraction of nonerased points. However, instead of checking whether the selected point can be found, as in our monotonicity tester, the convexity tester checks a more complicated "goodness condition" in each visited interval of the binary search tree. It boils down to checking that the slope of the functions between pairs of carefully selected points satisfies the convexity condition. In addition to spending queries on erased points due to sampling, like in the monotonicity tester, our tester also performs "walking queries" to find the nearest nonerased points to the left and to the right of the pivots in our random binary search tree. We show that the overhead in the query complexity due to querying erased points is only a factor of $O(1)$.

3.4 GENERIC TRANSFORMATIONS

In this section, we present our transformations that make two classes of testers erasure-resilient. Our transformations apply to (1) Proximity Oblivious Testers

(POTs) that are additionally restricted to be sample-based (Theorem 3.4.3), and (2) sample-based testers for extendable properties (Theorem 3.4.8).

Recall that a tester is called sample-based if its queries are distributed uniformly and independently at random. A 1-sided error sample-based tester always accepts functions that satisfy the specified property. Sample-based testers were first considered by Goldreich et al. (1998) and systematically studied by Goldreich & Ron (2016) and Fischer et al. (2015). In particular, Goldreich & Ron (2016); Fischer et al. (2015) show that certain types of query-based testers yield sample-based testers with sublinear (but dependent on the size of the input) sample complexity.

3.4.1 Sample-Based Proximity Oblivious Testers

In this section, we give a simple transformation that makes every POT that queries uniformly and independently random domain points erasure-resilient. POTs were defined by Goldreich & Ron (2011) and further studied by Goldreich & Kaufman (2011) and Goldreich & Shinkar (2016). We first define POTs.

Definition 3.4.1 (POT, Goldreich & Shinkar (2016))**.** *Let $\mathcal{P}$ be a property, let $\rho : (0,1] \to (0,1]$ be a monotone function and let $c \in (0,1]$ be a constant. A tester T is a (ρ, c)-*POT *for $\mathcal{P}$ if*

- *for every function $f \in \mathcal{P}$, the probability that T accepts f is at least c, and*
- *for every function $f \notin \mathcal{P}$, the probability that T accepts f is at most $c - \rho(\varepsilon_f)$, where ε_f denotes the relative Hamming distance of f to $\mathcal{P}$.*

It is important to note that POTs in general can query arbitrary domain points. Next, we define sample-based POTs, a restriction of POTs.

Definition 3.4.2 (Sample-based POT)**.** *A* POT *whose queries are distributed uniformly and independently at random is called a sample-based* POT*.*

Erasure-resilient versions of POTs and sample-based POTs can be defined analogously. Next, we state our generic transformation for sample-based POTs.

Theorem 3.4.3. *If T is a sample-based (ρ,c)-*POT *for a property $\mathcal{P}$ that makes q queries, then there exists a sample-based α-erasure-resilient (ρ',c)-*POT *T' for $\mathcal{P}$ that makes q queries for all $\alpha < \rho(\varepsilon_f)/q$, where $\rho'(x) = \rho(x) - \alpha \cdot q$ for $x \in (0,1]$.*

Proof. Let $\mathcal{P}$ be a property of functions over a domain $\mathcal{D}$. The tester T' queries q uniform and independent points from $\mathcal{D}$. It accepts if the sample has an erased point. Otherwise, it runs T on the q sampled nonerased points and accepts if and only if T accepts.

Consider an α-erased function $f \in \mathcal{P}$ and a completion $f^r \in \mathcal{P}$. The tester T accepts f^r with probability at least c. If T accepts f^r on querying a sample $S \subseteq \mathcal{D}$, then T' also accepts f on S. Thus, the probability that T' accepts f is at least c.

A tuple $W \in \mathcal{D}^q$ is a *witness* for a function $g \notin \mathcal{P}$, if T rejects g upon sampling W. Consider an α-erased function $f \notin \mathcal{P}$. Every completion f^r of f is ε_f-far from $\mathcal{P}$. Since T rejects f^r with probability at least $1-c+\rho(\varepsilon_f)$, at least $(1-c+\rho(\varepsilon_f)) \cdot |\mathcal{D}|^q$ tuples in $\mathcal{D}^q$ are witnesses for f^r. Erasing one point can affect at most $q \cdot |\mathcal{D}|^{q-1}$ witnesses. Thus, erasing an α fraction of points can affect at most $\alpha \cdot q \cdot |\mathcal{D}|^q$ witnesses. At least $(1 - c + \rho(\varepsilon_f) - \alpha \cdot q) \cdot |\mathcal{D}|^q$ out of $|\mathcal{D}|^q$ tuples are witnesses with no points (in them) erased. The probability that T' samples such a tuple (and rejects f) is at least $1 - c + \rho(\varepsilon_f) - \alpha \cdot q = 1 - c + \rho'(\varepsilon_f)$. Hence, the probability that T' accepts f is at most $c - \rho'(\varepsilon_f)$. This probability is less than c for all $\alpha < \rho(\varepsilon_f)/q$. □

Low degree univariate polynomials. We apply Theorem 3.4.3 to a POT designed by Rubinfeld & Sudan (1996) for the property of being a univariate polynomial of degree at most d over a finite field $\mathbb{F}$ and get an α-erasure-resilient ε-tester for this property. Consider a function $f : \mathbb{F} \to \mathbb{F}$ that we would like to test for being a univariate polynomial of degree at most d. The tester by Rubinfeld & Sudan (1996) selects $d+2$ points uniformly and independently at random from $\mathbb{F}$ and checks whether there is a univariate polynomial of degree at most d that fits all these points (by interpolation). It accepts if there is such a polynomial and rejects otherwise. Call this tester T. It is evident that T always accepts univariate polynomials of degree at most d. Rubinfeld & Sudan (1996) also prove that T rejects with probability at least ε_f if f is not a univariate polynomial of degree at most d, where ε_f denotes the distance of f from the property. Therefore, T is a sample-based $(\rho, 1)$-POT for this property, where ρ is the identity function. By Theorem 3.4.3, there exists a sample-based α-erasure-resilient $(\rho', 1)$-POT, say T', that makes $d+2$ queries, where $\rho'(x) = x - \alpha \cdot (d+2)$. The probability that T' rejects an α-erased function f that is ε-far from univariate polynomials of degree at most d is at least $\varepsilon - \alpha \cdot (d+2)$. The corollary follows.

Corollary 3.4.4. *There exists a sample-based α-erasure-resilient ε-tester for the property of being a univariate polynomial of degree at most d over a finite field $\mathbb{F}$ that works for all $\alpha \in [0,1), \varepsilon \in (0,1)$, where $\alpha+\varepsilon < 1$ and $\alpha < \frac{\varepsilon}{d+2}$, with query complexity $O\left(\frac{d+2}{\varepsilon-\alpha(d+2)}\right)$.*

3.4.2 Sample-Based Testers for Extendable Properties

In this section, we define extendable properties and present our generic transformation for sample-based testers of such properties. First, we define the extension of a function.

Definition 3.4.5 (Extension of a Function)**.** *Given $\mathcal{S} \subseteq \mathcal{T}$, the extension of a function $f : \mathcal{S} \to \mathcal{R}$ to a domain $\mathcal{T}$ is a function $g : \mathcal{T} \to \mathcal{R}$ that agrees with f on every point in $\mathcal{S}$.*

Our definition of extendable properties is a generalization of the notion of *edge-*

transitive properties that allow extension by Jha & Raskhodnikova (2013). A property is edge-transitive if it is fully characterized by predicates on pairs of domain points. Such a property allows extension if every function that satisfies the property on a subdomain can be extended to one that satisfies the property on the whole domain. We now define extendable properties.

Definition 3.4.6 (Extendable Property)**.** *For a domain $\mathcal{D}$ and all $\mathcal{S} \subseteq \mathcal{D}$, let $\mathcal{P}_\mathcal{S}$ denote a set of functions over domain $\mathcal{S}$. The property $\bigcup_{\mathcal{S} \subseteq \mathcal{D}} \mathcal{P}_\mathcal{S}$ is extendable if, for all $\mathcal{S}, \mathcal{T} : \mathcal{S} \subseteq \mathcal{T} \subseteq \mathcal{D}$,*

1. *for every function $f \in \mathcal{P}_\mathcal{S}$, there is an extension $f' \in \mathcal{P}_\mathcal{T}$, and*
2. *for every function $f \notin \mathcal{P}_\mathcal{S}$, every function $f' \in \mathcal{P}_\mathcal{T}$ is such that the Hamming distance of f from $f'_\mathcal{S}$ is at least the Hamming distance of f from $\mathcal{P}_\mathcal{S}$.*

We now give examples of some properties that are extendable and some that are not. A lot of properties that we deal with, in this chapter, are edge-transitive properties that allow extension. Monotonicity over arbitrary partial orders, (α, β)-Lipschitz properties over hypergrids, and more generally BDPs over hypergrids are all edge-transitive properties that allow extension. However, the class of properties for which we are able to design erasure-resilient testers is a strict superset of edge-transitive properties that allow extension. One such property is convexity of real-valued functions over the domain $[n]$. Another one is the property of Boolean functions over $[n]$ whose value alternates between 0 and 1 at most k times when moving from domain point 1 to domain point n. Both properties are not edge-transitive, but are extendable.

We now describe two properties defined over $[n]$ that are not extendable. The first among these, denoted $\mathcal{P}'$, is equal to $\bigcup_{I \subseteq [n]} \mathcal{P}'_I$, where $\mathcal{P}'_I$ for $I \subseteq [n]$ is the set of all integer-valued functions that are strictly increasing w.r.t. the ordering on points in I. The property $\mathcal{P}'$ is not extendable, since it does not satisfy the first condition in Definition 3.4.6. To see this, consider the function $f : \{1,3\} \to \mathbb{Z}$ such that $f(1) = 1, f(3) = 2$. Clearly, f belongs to $\mathcal{P}'_{\{1,3\}}$ and hence to $\mathcal{P}'$. But f cannot be extended to $\{1,2,3\}$ while satisfying $\mathcal{P}'$. The second property, denoted $\hat{\mathcal{P}}$, is equal to $\bigcup_{I\subseteq[n]} \hat{\mathcal{P}}_I$. A function $f : I \to [n]$ is in $\hat{\mathcal{P}}_I$ for $I \subseteq [n]$, if for each $i \in I$, we have $f(i) \leq |\{j \in I : j \leq i\}|$. In other words, a function satisfies $\hat{\mathcal{P}}_I$ if the value of the function at each point i is at most the number of points $j \leq i$ where it is defined. It is easy to see that this property satisfies the first condition in Definition 3.4.6. We will show that it does not satisfy the second condition. Consider the function f defined on $\{1,3\}$, where $f(1) = 1$ and $f(3) = 3$. This function is 1/2-far from $\hat{\mathcal{P}}_I$. But this can be extended to the function g over $[n]$ as $g(i) = i$ for all $i \in [n]$. Clearly, g satisfies $\hat{\mathcal{P}}$, violating the second condition in Definition 3.4.6.

We are now ready to describe our generic transformation for extendable properties. In what follows, we will be talking about functions defined over various domains, their extensions, and completions[1]. The next lemma is used in the proof of our generic transformation.

Lemma 3.4.7. *Let $\bigcup_{\mathcal{S}\subseteq\mathcal{D}} \mathcal{P}_{\mathcal{S}}$ be an extendable property. Let $\alpha \in [0,1), \varepsilon \in (0,1)$ such that $\alpha + \varepsilon < 1$. Consider an α-erased function f over domain $\mathcal{D}$ and let $\mathcal{N} \subseteq \mathcal{D}$ be the set of nonerased points in it. If $f \in \mathcal{P}_{\mathcal{D}}$, then $f_{|\mathcal{N}} \in \mathcal{P}_{\mathcal{N}}$. If f is ε-far from $\mathcal{P}_{\mathcal{D}}$, then $f_{|\mathcal{N}}$ is $\frac{\varepsilon}{1-\alpha^*}$-far from $\mathcal{P}_{\mathcal{N}}$, where $\alpha^* \leq \alpha$ is the true fraction of erasures in f.*

[1]When we say that $g : \mathcal{T} \to \mathcal{R}$ is an extension of $f : \mathcal{S} \to \mathcal{R}$, we mean that g and f are functions defined on different domains $\mathcal{T}$ and $\mathcal{S}$ and that $\mathcal{S} \subseteq \mathcal{T}$. The function f is *not defined* on the set $\mathcal{T} \setminus \mathcal{S}$ and $f(x) = g(x)$ for all $x \in \mathcal{S}$. The functions f and/or g may or may not have some of their function values erased. On the other hand, when we say that g is a completion of f, it must be the case that f and g are functions defined on the same domain, say $\mathcal{T}'$, with f being (possibly) erased, but not undefined, on some points in $\mathcal{T}'$. Also, the values of f and g must be equal on the points in $\mathcal{T}'$ where f is nonerased.

Proof. Suppose that $f \in \mathcal{P}_{\mathcal{D}}$. Let $f_* \in \mathcal{P}_{\mathcal{D}}$ be a completion of f. As f_* is a function defined over $\mathcal{D}$ and f_* has no erasures, f_* is an extension of $f_{|\mathcal{N}}$. Therefore, $f_{|\mathcal{N}} \in \mathcal{P}_{\mathcal{N}}$, since $\bigcup_{\mathcal{S}\subseteq\mathcal{D}} \mathcal{P}_{\mathcal{S}}$ is an extendable property.

Now, suppose that f is ε-far from $\mathcal{P}_{\mathcal{D}}$. Then, every completion of f needs to be changed in at least an $\varepsilon|\mathcal{D}|$ points in $\mathcal{N}$ to satisfy $\mathcal{P}_{\mathcal{D}}$. Assume for the sake of contradiction that the relative Hamming distance of $f_{|\mathcal{N}}$ to $\mathcal{P}_{\mathcal{N}}$ is $\varepsilon' < \frac{\varepsilon|\mathcal{D}|}{|\mathcal{N}|} = \frac{\varepsilon}{1-\alpha^*}$. Let g be the function in $\mathcal{P}_{\mathcal{N}}$ closest to $f_{|\mathcal{N}}$. Let g^e be an extension of g to $\mathcal{D}$ that satisfies $\mathcal{P}_{\mathcal{D}}$. Define an extension of $f_{|\mathcal{N}}$ to $\mathcal{D}$, say f^e as follows. The function f^e takes the same values as $f_{|\mathcal{N}}$ on points in $\mathcal{N}$ and takes the same values as g^e on the remaining points. Note that f^e is a completion of f as well. Clearly, f^e can be made to satisfy $\mathcal{P}_{\mathcal{D}}$ by changing its values on $\varepsilon'|\mathcal{N}| < \varepsilon|\mathcal{D}|$ many points in $\mathcal{N}$, which contradicts the assumption that f is ε-far from $\mathcal{P}_{\mathcal{D}}$. $\square$

Our generic transformation for sample-based testers of extendable properties follows.

Theorem 3.4.8. *Let $q(\cdot,\cdot)$ be a function that is nondecreasing in the first argument and nonincreasing in the second argument. Let $\bigcup_{\mathcal{S}\subseteq\mathcal{D}} \mathcal{P}_{\mathcal{S}}$ be an extendable property. Let $\varepsilon \in (0,1)$. Suppose T is a 1-sided error sample-based tester for the property $\bigcup_{\mathcal{S}\subseteq\mathcal{D}} \mathcal{P}_{\mathcal{S}}$, such that for every $\mathcal{S} \subseteq \mathcal{D}$, the tester T makes $q(|\mathcal{S}|,\varepsilon)$ queries from $\mathcal{S}$ to ε-test for $\mathcal{P}_{\mathcal{S}}$. Assume also that for every $\mathcal{S} \subseteq \mathcal{D}$, the probability that T tests $\mathcal{P}_{\mathcal{S}}$ correctly does not decrease when it makes more queries. Then, for all $\alpha \in [0,1)$ such that $\alpha+\varepsilon < 1$, the property $\mathcal{P}_{\mathcal{D}}$ can be α-erasure-resiliently ε-tested with 1-sided error using $O\left(\frac{q(|\mathcal{N}|,\frac{\varepsilon}{1-\alpha^*})}{1-\alpha^*}\right)$ independent and uniformly random queries, where $\alpha^* \leq \alpha$ is the true fraction of erasures in the function being tested.*

Proof. Consider a "thought tester" T' that, given oracle access to an α-erased function f, samples $Q = 2q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*})/(1-\alpha^*)$ points uniformly and independently at random from $\mathcal{D}$, where $\mathcal{N}$ denotes the set of nonerased points in f and α^* is the true fraction of erasures in f. If there are fewer than $q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*})$ nonerased points in the sample, T' accepts. Otherwise, it runs T on the sampled nonerased points and accepts if and only if T accepts.

The expected number of nonerased points in a uniform sample of size Q from $\mathcal{D}$ is at least $Q \cdot (1-\alpha^*) = 2q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*})$. By the Chernoff bound, the probability that T' samples fewer than $q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*})$ nonerased points is at most $e^{-q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*})/4}$.

If $f \in \mathcal{P}$, then $f_{|\mathcal{N}} \in \mathcal{P}_{\mathcal{N}}$ by Lemma 3.4.7, and the tester T' always accepts $f_{|\mathcal{N}}$. Assume now that f is ε-far from $\mathcal{P}$. Then $f_{|\mathcal{N}}$ is $\frac{\varepsilon}{1-\alpha^*}$-far from $\mathcal{P}_{\mathcal{N}}$ by Lemma 3.4.7. Therefore T rejects $f_{|\mathcal{N}}$ with probability at least $2/3$ on a sample of size at least $q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*})$. Thus, by a union bound, the probability that T' accepts is at most $1/3 + e^{-q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*})/4}$. This probability can be brought below $1/3$ by repeating T' a small constant number of times, whenever $q(|\mathcal{N}|, \frac{\varepsilon}{1-\alpha^*}) \geq 8$. □

In the following, we show a few applications of Theorem 3.4.3.

Convexity of Images. A black and white image, represented by a function $f : S \to \{0,1\}$ for a subset S of $[n]^2$, is convex if and only if for every pair of points $u, v \in S$ such that $f(u) = f(v) = 1$, every point $t \in S$ on the line joining u and v satisfy $f(t) = 1$. Convexity is an extendable property. Testing whether an image, represented by a function $f : [n]^2 \to \{0,1\}$, is convex has been studied by Berman et al. (2016b). They give a 1-sided error sample-based ε-tester for this property that makes $O(1/\varepsilon^{4/3})$ uniform queries. Their proofs go through even if the domain of f is an arbitrary subset of $[n]^2$. The corollary now follows by applying Theorem 3.4.8 to the tester of Berman et al. (2016b).

Corollary 3.4.9. *There exists a sample-based α-erasure-resilient ε-tester for convexity of black and white images that works for all $\alpha \in [0,1), \varepsilon \in (0,1/2)$, where*

$\alpha + \varepsilon < 1$, and has query complexity $O\left(\frac{1}{\varepsilon^{4/3}}\right)$.

Proof. The proof of Theorem 3.4.8 guarantees a thought tester for convexity that uses $O\left(\frac{1}{(\frac{\varepsilon}{1-\alpha^*})^{4/3}}\right)$ independent and uniformly distributed queries. A sample-based tester making $O\left(\frac{1}{\varepsilon^{4/3}}\right)$ queries (and using the same decision rule) also gives the same guarantees as the thought tester, since making more queries cannot decrease the correctness probability. $\square$

Monotonicity over Poset Domains. A real-valued function f defined on a partially ordered domain is monotone if the function values respect the order relation of the poset. Monotonicity is an extendable property. The tester by Fischer et al. (2002) samples $O(\sqrt{N/\varepsilon})$ points uniformly at random and checks for violations to monotonicity among them. The corollary follows by applying Theorem 3.4.8 to this tester.

Corollary 3.4.10. *There exists a sample-based α-erasure-resilient ε-tester for monotonicity of real-valued functions over N element posets that works for all $\alpha \in [0,1), \varepsilon \in (0,1)$, where $\alpha + \varepsilon < 1$, and has query complexity $O\left(\sqrt{\frac{N}{\varepsilon}}\right)$.*

Proof. The proof of Theorem 3.4.8 guarantees a thought tester for monotonicity that makes $O\left(\sqrt{\frac{N(1-\alpha^*)}{\frac{\varepsilon}{1-\alpha^*}}}\right)$ independent and uniformly distributed queries. A sample-based tester making $O\left(\sqrt{\frac{N}{\varepsilon}}\right)$ queries (and using the same decision rule) also gives the same guarantees as the thought tester, since making more queries cannot decrease the correctness probability. $\square$

Boolean Functions with k Runs. A function $f : [n] \to \{0,1\}$ has k *runs* if the list $f(1), f(2), \ldots, f(n)$ has at most $k-1$ alternations of values. The problem is to test whether a given function $f : [n] \to \{0,1\}$ has k runs or is ε-far from this property. Kearns & Ron (2000) studied a relaxation of this problem. Specifically, they showed that $O(1/\varepsilon^2)$ queries suffice to test whether a Boolean function has k runs or is ε-far from being a k/ε-run function. They also developed a sample-based $O(\sqrt{k}/\varepsilon^{2.5})$-query tester for this relaxation and proved that every sample-based ε-tester for the k-run property requires $\Omega(\sqrt{k})$ queries. Balcan et al. (2012) obtained a $O(1/\varepsilon^4)$-query tester for this property in the active testing model. They also developed a sample-based $O(\sqrt{k}/\varepsilon^6)$-query tester[2]. Canonne et al. (2017) designed a 1-sided error nonadaptive ε-tester for Boolean k-run functions that works for all $k \in [n], \varepsilon \in (0,1)$, with query complexity $O(\frac{k+1}{\varepsilon})$. In addition, they also show a two-sided nonadaptive ε-tester for Boolean k-run functions that works for all $k \in [n], \varepsilon \in (0,1)$ with query complexity $\tilde{O}(1/\varepsilon^7)$. We show the following.

Theorem 3.4.11. *There exists a sample-based ε-tester for the property of being a Boolean function with k runs over $[n]$ that works for all $\varepsilon \in (\frac{k^2}{n}, 1)$, with query complexity $O\left(\min\left\{\frac{k\log k}{\varepsilon}, \frac{\sqrt{k}}{\varepsilon^6}\right\}\right)$.*

Algorithm 3.1 Tester for k-run Boolean functions

Input: parameters $k \in \mathbb{N}, \varepsilon \in (\frac{k^2}{n}, 1)$; oracle access to $f : [n] \to \{0,1\}$

```
1: Query the values at $\frac{3(k+1)\cdot\log(k+1)}{\varepsilon}$ points uniformly and independently at random.
2: Reject if the values of $f$ at these points alternate $k$ or more times with respect
   to the ordering on the domain; accept otherwise.
```

Our tester for being a k-run function is given in Algorithm 3.1. It always accepts a function f that has at most k runs. The following lemma implies Theorem 3.4.11.

Lemma 3.4.12. *If f is ε-far from being a k-run function, Algorithm 3.1 rejects with probability at least $2/3$.*

[2]Both Kearns & Ron (2000) and Balcan et al. (2012) study Boolean functions over $[0,1]$. We note that their algorithms also work for Boolean functions over $[n]$.

Proof. For $j \in [n]$ and $b \in \{0,1\}$, let $T_{b,j}$ denote the set consisting of the smallest $\lceil n \cdot \varepsilon/(k+1) \rceil$ points in the set $\{x : j \leq x \leq n \text{ and } f(x) = b\}$, that is, the set of points between j and n where f takes the value b. For a set $S \subseteq [n]$, let $\max(S)$ denote the largest element in S. We will first describe a process to construct a few disjoint subsets of $[n]$ with some special properties.

- Let $S_1 = T_{b,1}$ such that $\max(T_{b,1}) < \max(T_{1-b,1})$.
- For $i \geq 2$, the sets S_i are defined as follows. Let the value that f takes on the elements in S_{i-1} be b and let $j = \max(S_{i-1})$. Set $S_i = T_{1-b,j+1}$. Stop if $\max(S_i) = n$ or $S_i = \emptyset$.

The sets that this process constructs have the following properties. All S_i's are subsets of $[n]$. Each point in S_{i+1} is larger than every point in S_i for all i. The function f takes the same value on all points in S_i for all i. The value of f on points in S_{i+1} is the complement of the value of f on points in S_i for all i.

Next, we show that our process constructs sets $S_1, S_2, \ldots S_{k+1}$ each of size $\lceil n \cdot \varepsilon/(k+1) \rceil$, if f is ε-far from satisfying the property. Let the process construct nonempty sets $S_1, S_2, \ldots S_t$. Assume for the sake of contradiction that $t \leq k$. Let $S'_1 = \{x : 1 \leq x \leq \max(S_1)\}$. Let $S'_i = \{x : \max(S_{i-1}) < x \leq \max(S_i)\}$ for all $1 < i \leq t$. Note that for all $i \in [t]$, if f takes the value b on elements in S_i, then f takes the value $1-b$ on elements in $S'_i \setminus S_i$. We will describe a function f' that has at most t runs. Set the values of f' on each $x \in S'_1 \setminus S_1$ to the value that f takes on S_1. For each $1 < i \leq t$, set the values of f' on S_i to the value of f on $S'_i \setminus S_i$. On the rest of the points, f' takes the same value as f. We will now show that f' has at most t alternating intervals. The function f' takes the same value on points in $S'_1 \cup S'_2$. Also, for each $1 < i \leq t$, the function f' is constant on S'_i. Thus, f' has at most t runs. Also, f' differs from f in at most $t \cdot \lceil n \cdot \varepsilon/(k+1) \rceil \leq k \cdot \lceil n \cdot \varepsilon/(k+1) \rceil \leq n\varepsilon$

points, for $k < \sqrt{n\varepsilon}$. This is a contradiction.

Using the fact that $k+1$ such subsets exist, we show that the tester will detect a violation with high probability. For a particular i, the probability that none of the points selected by the algorithm lie in S_i is at most

$$(1-\varepsilon/(k+1))^{3(k+1)\log(k+1)/\varepsilon} \leq 1/(k+1)^3.$$

Therefore, by a union bound, the probability that there exists an i such that none of the points selected by the algorithm lies in S_i is at most $(k+1)^{-2} < 1/3$ for $k \geq 1$. □

Since the property of being a k-run function is extendable, applying Theorem 3.4.8 to Theorem 3.4.11 yields the following corollary.

Corollary 3.4.13. *There exists a sample-based α-erasure-resilient ε-tester for the property of being a k-run Boolean function over $[n]$, that works for all $\alpha \in [0,1)$, $\varepsilon \in \left(\frac{k^2}{n}, 1\right)$, where $\alpha + \varepsilon < 1$, with query complexity $O\left(\min\left\{\frac{k \cdot \log k}{\varepsilon}, \frac{\sqrt{k}}{\varepsilon^6}\right\}\right)$.*

3.5 ERASURE-RESILIENT MONOTONICITY TESTER FOR THE LINE

In this section, we prove Theorem 3.3.1. Recall that, for a function $f : [n] \to \mathbb{R} \cup \{\perp\}$, the set of nonerased points (the ones that map to $\mathbb{R}$) is denoted by $\mathcal{N}$. The function f is monotone if $x < y$ implies $f(x) \leq f(y)$ for all $x, y \in \mathcal{N}$. Given a function $f : [n] \to \mathbb{R} \cup \{\perp\}$ that is not monotone, a *violation to monotonicity* of f is a pair of points $x, y \in \mathcal{N}$ such that $x < y$ and $f(x) > f(y)$. The points x and y are said to *violate the monotonicity* of f.

We present our tester in Algorithm 3.2. It has oracle access to $f : [n] \to \mathbb{R} \cup \{\perp\}$ and takes α, ε and n as inputs. The tester knows neither the set $\mathcal{N}$ nor the value of

$|\mathcal{N}|$ in advance. However, it gets a lower bound on $|\mathcal{N}|$ in the form of $n(1-\alpha)$. In each iteration, it performs a randomized binary search for a nonerased index sampled uniformly at random (u.a.r.) from $\mathcal{N}$ and rejects if it finds a violation to monotonicity. In the description of our tester, we use $I[i,j]$ to denote the set of natural numbers from i until and including j. We also refer to $I[i,j]$ as the interval from i to j.

Algorithm 3.2 Erasure-Resilient Monotonicity Tester for the Line

```
Input: parameters $\alpha \in [0,1), \varepsilon \in (0,1)$; oracle access to $f : [n] \to \mathbb{R} \cup \{\perp\}$
 1: Set $Q = \lceil \frac{60 \log n}{\varepsilon} \rceil$.
 2: Accept at any point if the number of queries exceeds $Q$.
 3: repeat $\frac{2}{\varepsilon}$ times:
 4:     Sample points uniformly at random from $I[1,n]$ and query them until we get
    a point $s \in \mathcal{N}$.
 5:     Set $\ell \leftarrow 1$, $r \leftarrow n$.
 6:     while $\ell \leq r$ do
 7:         Sample points uniformly at random from $I[\ell,r]$ and query them until we
    get a point $m \in \mathcal{N}$.
 8:         if $s < m$ then set $r \leftarrow m-1$ and Reject if $f(s) > f(m)$.
 9:         if $s > m$ then set $\ell \leftarrow m+1$ and Reject if $f(s) < f(m)$.
10:         if $s = m$ then Go to Step 3.                    ▷ Search completed.
11: Accept.
```

One of the key ideas in our analysis is to view each iteration of the loop in Step 3 as first sampling a binary search tree $\mathcal{T}$ on $\mathcal{N}$ according to a particular distribution $\mathcal{D}_{\mathcal{T}}$, and then traversing a uniformly random rooted path in that tree, where a rooted path refers to a path from the root of a tree to an arbitrary node in that tree. This view enables us to prove the correctness of the tester by generalizing an argument due to Ergün et al. (2000) for the case when Algorithm 3.2 manages to complete all iterations of Step 3 before it runs out of queries. The challenge is that the algorithm might get stuck pursuing long paths in the search tree and waste many queries on erased points. To resolve the issue of many possible queries to erased points, we prove an upper bound on the expected number of queries made while traversing a uniformly random rooted path in a binary search tree sampled from $\mathcal{D}_{\mathcal{T}}$. We combine this with the fact that the expected depth of a binary search tree sampled from $\mathcal{D}_{\mathcal{T}}$ is $O(\log n)$, in order to obtain the final bound on the probability that the algorithm exceeds its query budget (and wrongly accepts functions that are far from monotone).

Analysis

We analyze the tester in this section. The query complexity of the tester is clear from its description. The main statement of Theorem 3.3.1 follows from Lemma 3.5.1, proved next.

Lemma 3.5.1. *Algorithm 3.2 accepts if f is monotone, and rejects with probability at least $2/3$ if f is ε-far from monotone.*

Proof. The tester accepts whenever f is monotone. To prove the other part of the lemma, assume that f is ε-far from monotone. Let $\alpha^* \leq \alpha$ denote the actual fraction of erasures in f. That is, $|\mathcal{N}| = n(1-\alpha^*)$. Consider a 'thought tester' that is identical to Algorithm 3.2 except that it repeats Step 3 only $\frac{2(1-\alpha^*)}{\varepsilon}$ times. Since $\frac{2(1-\alpha^*)}{\varepsilon} \leq \frac{2}{\varepsilon}$, the probability that Algorithm 3.2 accepts is at most the probability that the 'thought tester' accepts. In what follows, we analyze this 'thought tester' instead of Algorithm 3.2.

Let A be the event that the tester accepts f. Let q denote the total number of queries made. We prove that $\Pr[A] \leq 1/3$. The event A occurs if either $q > Q$ or the tester does not find a violation in any of the $2(1-\alpha^*)/\varepsilon$ iterations of Step 3. Thus, $\Pr\left[A\right] \leq \Pr\left[A \mid q \leq Q\right] + \Pr\left[q > Q\right]$.

Each iteration of Step 3 can be viewed as traversing a uniformly random search path in a binary search tree on $\mathcal{N}$, where the tree is sampled from a particular distribution. More formally, we describe a two-stage random process that provides such an alternate view of a single iteration of Step 3.

The first stage of the process involves constructing a binary search tree $\mathcal{T}$ over $\mathcal{N}$ as follows. Each node of $\mathcal{T}$ is associated with an interval $I[a, b]$ and has a nonerased point from $I[a, b]$ set as its key, where $1 \leq a \leq b \leq n$. The root node of $\mathcal{T}$ is associated with the interval $I[1, n]$. Consider an arbitrary node Γ of the tree whose key has not been set yet. Let $I[a, b]$ be the interval associated with it. Repeatedly sample points u.a.r. from $I[a, b]$ and query them, until we get a point $p \in \mathcal{N} \cap I[a, b]$. Set the key of the node Γ to p. If p is the *leftmost* nonerased point in $I[a, b]$, then the node Γ does not have a left child and its right child is associated with $I[p + 1, b]$. Similarly, if p is the *rightmost* nonerased point in $I[a, b]$, then Γ does not have a right child and its left child is associated with $I[a, p - 1]$. Otherwise, we associate the intervals $I[a, p - 1]$ and $I[p+1, b]$ with the left and right children of Γ, respectively. We use $\mathcal{D}_{\mathcal{T}}$ to denote the distribution on binary search trees sampled in this way. Note that the leaves of $\mathcal{T}$ are nodes associated with intervals containing exactly one nonerased point. Since the key of each node in $\mathcal{T}$ is a unique nonerased point in $\mathcal{N}$, we will henceforth refer to the nodes of $\mathcal{T}$ using their keys.

In the second stage of the random process, we sample a node $s \in \mathcal{N}$ of $\mathcal{T}$ u.a.r. and reject if s with any one of its ancestors in $\mathcal{T}$ violate the monotonicity of f. In other words, we sample a uniformly random rooted path from $\mathcal{T}$ and check whether the deepest node on that path violates the monotonicity of f with any of its ancestors.

We now argue that each iteration of Step 3 of our algorithm simulates the above random process. Consider the search path traversed by the algorithm in an iteration. It is easy to see that there exists a binary search tree $\mathcal{T}$ sampled from $\mathcal{D}_{\mathcal{T}}$ such that $\mathcal{T}$ contains the search path traversed by the algorithm. Each node of this binary search tree $\mathcal{T}$ is a unique element of $\mathcal{N}$. As the algorithm samples its search point

s u.a.r. from $\mathcal{N}$, the node corresponding to s in $\mathcal{T}$ is a node sampled u.a.r. from the tree $\mathcal{T}$. The algorithm checks whether the monotonicity of f is violated by s and any of the nonerased points on its search path. This is exactly the same as checking for violations to monotonicity of f between s and its ancestors in $\mathcal{T}$. The number of queries made to the intervals associated with the nodes along the path from the root of $\mathcal{T}$ to s during the random process has the same distribution as the number of queries made by the tester while traversing the search path to s.

We are now ready to bound the probability that the tester does not reject in an iteration of Step 3, conditioned on the event that $q \leq Q$. Consider a binary search tree $\mathcal{T}$ over $\mathcal{N}$ sampled from the distribution $\mathcal{D}_{\mathcal{T}}$. A point $s \in \mathcal{N}$ is called *searchable* with respect to $\mathcal{T}$ if s does not violate the monotonicity of f with any of its ancestors in $\mathcal{T}$. Consider two points $i, j \in \mathcal{N}$, where $i < j$, both searchable with respect to $\mathcal{T}$. Let $a \in \mathcal{N}$ be the lowest common ancestor of the nodes i and j in $\mathcal{T}$. Since i and j are both searchable, it must be the case that $f(i) \leq f(a)$ and $f(a) \leq f(j)$ and hence, $f(i) \leq f(j)$. Thus, for every tree $\mathcal{T}$ sampled from $\mathcal{D}_{\mathcal{T}}$, the function f, when restricted to the domain points that are searchable with respect to $\mathcal{T}$, is monotone. Therefore, if f is ε-far from monotone, for every binary search tree $\mathcal{T}$ sampled from $\mathcal{D}_{\mathcal{T}}$, at least εn points from $\mathcal{N}$ are not searchable. That is, at least an $\frac{\varepsilon}{1-\alpha^*}$ fraction of the points in $\mathcal{N}$ are not searchable with respect to $\mathcal{T}$. Thus, the random process, and each iteration of Step 3 of the tester, reject with probability at least $\frac{\varepsilon}{1-\alpha^*}$. Consequently,

$$\Pr\left[A \mid q \leq Q\right] \leq \left(1 - \frac{\varepsilon}{1-\alpha^*}\right)^{\frac{2(1-\alpha^*)}{\varepsilon}} \leq e^{-2} < \frac{1}{6}.$$

In the rest of the proof, we bound $\Pr[q > Q]$. We first prove an upper bound on the expected number of queries to traverse a uniformly random rooted path in a binary search tree $\mathcal{T}$ sampled according to $\mathcal{D}_{\mathcal{T}}$. Recall that a rooted path in a search tree $\mathcal{T}$ is a path from the root to some node in $\mathcal{T}$. Let I be the interval associated with a node Γ of $\mathcal{T}$ and let α_I denote the fraction of erased points in I. The number of queries needed to sample a nonerased point from I with uniform sampling is a geometric random variable with expectation $1/(1-\alpha_I)$. We define the *query-weight* of node Γ to be this expectation. The query-weight of a path in $\mathcal{T}$ is the sum of query-weights of the nodes on the path (which is equal to the expected total number of queries made by the random process to all the intervals on that path while constructing $\mathcal{T}$).

Claim 3.5.2. *Consider a binary search tree $\mathcal{T}$ of height h over $\mathcal{N}$, sampled according to the distribution $\mathcal{D}_{\mathcal{T}}$. The expected query-weight of a uniformly random rooted path in $\mathcal{T}$ is at most $\frac{h}{1-\alpha^*}$.*

Proof. There are exactly $|\mathcal{N}|$ rooted paths in $\mathcal{T}$. Let S denote the sum of query-weights of all the rooted paths. The expected query-weight of a uniformly random rooted path in $\mathcal{T}$ is then equal to $S/|\mathcal{N}|$.

Consider a node Γ in $\mathcal{T}$ associated with an interval I. There are $|I|(1-\alpha_I)$ nonerased points in I. The paths from the root of $\mathcal{T}$ to the nodes corresponding to each of these nonerased points pass through Γ. Hence, the query-weight of Γ gets added to the query-weights of all those paths. Therefore, the total contribution of Γ towards S is $|I|$, since the query-weight of Γ is $1/(1-\alpha_I)$. Note that the intervals associated with nodes at the same level of $\mathcal{T}$ are disjoint from each other. Hence, the total contribution to S from all nodes on the same level of $\mathcal{T}$ is at most n. Therefore, the value of S is at most $n \cdot h$, where h is the depth of $\mathcal{T}$. Observe that this quantity is independent of the fraction of erasures α. Therefore, the expected query-weight of a search path is at most $n \cdot h/|\mathcal{N}|$, which is at most $h/(1-\alpha^*)$, since $|\mathcal{N}| = n \cdot (1-\alpha^*)$. □

Next, we bound the expected height of a tree $\mathcal{T}$ sampled from $\mathcal{D}_{\mathcal{T}}$. Consider the following random process that constructs a binary tree T on the set $S = [k]$. The root of T is associated with the set S. The key of an arbitrary node v in T associated with a set $S' \subseteq S$ is a uniformly random element $x \in S'$. The left child of v is associated with the set $\{y \in S' : y < x\}$ if this set is nonempty. The right child of v is associated with the set $\{y \in S' : y > x\}$ if this set is nonempty. A tree constructed in this way is called a random binary search tree on k nodes. We now state a fact on the expected height of a random binary search tree.

Claim 3.5.3 (Pittel (1984); Devroye (1986); Drmota (2003); Reed (2003))**.** *If H_k is the random variable denoting the height of a random binary search tree on k nodes, then $\mathbb{E}[H_k] \leq 5 \log k$.*

It is easy to see that the height of a tree $\mathcal{T}$ sampled from $\mathcal{D}_{\mathcal{T}}$ has the same distribution as the random variable $H_{|\mathcal{N}|}$, since the key associated with each node in $\mathcal{T}$ is a uniformly random nonerased point from the interval associated with that node. Hence, the expected depth of $\mathcal{T}$ is at most $5 \log(|\mathcal{N}|) \leq 5 \log n$. The corollary below follows immediately.

Corollary 3.5.4. *The expected total number of queries made to all the intervals of a uniformly random rooted path in a random binary search tree $\mathcal{T}$ on $\mathcal{N}$, sampled according to $\mathcal{D}_{\mathcal{T}}$, is at most $\frac{5 \log n}{1-\alpha^*}$.*

It follows from Corollary 3.5.4 that the expected number of queries made by Algorithm 3.2 in a single iteration is at most $5 \log n/(1-\alpha^*)$. Hence, by the linearity of expectation, the expected number of queries made by the tester over all its iterations, $\mathbb{E}[q]$, is at most $10(1-\alpha^*) \log n/(\varepsilon \cdot (1-\alpha^*)) = 10 \log n/\varepsilon$. Applying Markov's inequality to q, we can then see that

$$\Pr[q > Q] \leq \frac{1}{6}.$$

Therefore, the probability that the tester does not reject is

$$\Pr[A] \leq \Pr[A \mid a \leq Q] + \Pr[q > Q] < \frac{1}{6} + \frac{1}{6} = \frac{1}{3}.$$

This completes the proof of the lemma. $\square$

3.6 ERASURE-RESILIENT MONOTONICITY TESTERS FOR THE HYPERGRID

In this section, we present our erasure-resilient tester for monotonicity over hypergrid domains and prove the following theorem, which is a special case of Theorem 3.3.2. We present the erasure-resilient testers for general BDPs in Section 3.7.

Theorem 3.6.1. *There exists a* 1*-sided error α-erasure-resilient ε-tester for monotonicity of real-valued functions on the hypergrid $[n]^d$ that works for all $\alpha \in [0,1), \varepsilon \in (0,1)$, where $\alpha + \varepsilon < 1$ and $\alpha \leq \varepsilon/248d$, with query complexity $O(\frac{d \log n}{\varepsilon})$.*

Let $\mathcal{L}$ denote the set of all *axis-parallel lines* in the hypergrid, where an axis-parallel line is a set of n distinct points in $[n]^d$ that agree on all but one coordinate. Our monotonicity tester, which is described in Algorithm 3.3, samples a uniformly random *axis-parallel line* in each iteration of Step 2 and does a randomized binary search for a uniformly random nonerased point on that line (as in Algorithm 3.2). It rejects if and only if a violation to monotonicity is found within its query budget. To analyze the tester, we first state two important properties of a uniformly random axis-parallel line in Lemma 3.6.2 and Lemma 3.6.3, which we jointly call the erasure-resilient dimension reduction. The statements and proofs of more general versions of these lemmas, applicable to all BDPs, are given in Section 3.7.

Lemma 3.6.2 (Dimension reduction: distance)**.** *Let ε_f be the relative Hamming distance of an α-erased function $f : [n]^d \to \mathbb{R} \cup \{\perp\}$ from monotonicity. For an axis-parallel line $\ell \in \mathcal{L}$, let $f_\ell : [n] \to \mathbb{R} \cup \{\perp\}$ denote the restriction of f to ℓ and let ε_ℓ denote the relative Hamming distance of f_ℓ from monotonicity. Then*

$$\mathbb{E}_{\ell \sim \mathcal{L}}[\varepsilon_\ell] \geq \frac{\varepsilon_f}{4d} - \alpha.$$

Lemma 3.6.3 (Dimension reduction: fraction of erasures)**.** *Consider an α-erased function $f : [n]^d \to \mathbb{R} \cup \{\perp\}$. For $\ell \in \mathcal{L}$, let α_ℓ denote the fraction of erased points in ℓ. Then, for every $\eta \in (0, 1)$,*

$$\Pr_{\ell \sim \mathcal{L}}\left[\alpha_\ell > \frac{\alpha}{\eta}\right] \leq \eta.$$

Algorithm 3.3 Erasure-Resilient Monotonicity Tester for $[n]^d$

Input: parameters $\varepsilon \in (0, 1), \alpha \in \left[0, \frac{\varepsilon}{248d}\right]$; oracle access to $f : [n]^d \to \mathbb{R} \cup \{\perp\}$

```
1: Set $Q = \left\lceil \frac{1200d \cdot \log n}{\varepsilon} \right\rceil$.
2: repeat $\frac{12d}{\varepsilon - 4d\alpha}$ times:
3:     Sample a line $\ell \in \mathcal{L}$ uniformly at random.
4:     Sample points u.a.r. from $\ell$ and query them until we get a point $s \in \mathcal{N}$.
5:     Perform a randomized binary search for $s$ on $\ell$ as in Algorithm 3.2.
6:     Reject if any violation to monotonicity is found.
7: Accept at any point if the number of queries exceed $Q$.
```

The query complexity of the tester is evident from its description. We will now prove its correctness in the following lemma, which will then imply Theorem 3.6.1.

Lemma 3.6.4. *Algorithm 3.3 accepts if f is monotone, and rejects with probability at least $2/3$ if f is ε-far from monotone.*

Proof. The tester accepts if f is monotone.

Assume that f is ε-far from monotone. Let $\alpha^* \leq \alpha$ denote the exact fraction of erasures in f. As in the proof of Lemma 3.5.1, consider a 'thought tester' that is identical to Algorithm 3.3 in all respects except that Step 2 is repeated only $\frac{12d}{\varepsilon - 4d\alpha^*}$ times. Since $\frac{12d}{\varepsilon-4d\alpha^*} \leq \frac{12d}{\varepsilon-4d\alpha}$, the probability that Algorithm 3.3 does not find a violation to monotonicity is at most the probability that the 'thought tester' does not find a violation to monotonicity. In the rest, we analyze the 'thought tester'.

Let A denote the event that the tester does not find a violation to monotonicity in any of its iterations. If q denotes the total number of queries made by the tester,

$$\Pr[A] \leq \Pr[A|q \leq Q] + \Pr[q > Q].$$

Let t denote $\frac{12d}{\varepsilon-4d\alpha^*}$, which is the number of iterations of Step 2 of the tester. Let A_i denote the event that the tester does not find a violation to the monotonicity of f in its i-th iteration. For $\ell \in \mathcal{L}$, let f_ℓ denote f restricted to the line ℓ. Let ε_ℓ denote the relative Hamming distance of f_ℓ from monotonicity. Let E_ℓ denote the event that the tester samples the line ℓ in a particular iteration.

We then have,

$$\Pr[A_i|q \leq Q] \leq \sum_{\ell \in \mathcal{L}} (1 - \varepsilon_\ell) \Pr[E_\ell] = 1 - \mathbb{E}_{\ell \sim \mathcal{L}}[\varepsilon_\ell].$$

Using Lemma 3.6.2 and the fact that $\varepsilon_f \geq \varepsilon$, we have,

$$\mathbb{E}_{\ell \sim \mathcal{L}}[\varepsilon_\ell] \geq \frac{\varepsilon_f}{4d} - \alpha^* \geq \frac{\varepsilon}{4d} - \alpha^*.$$

Therefore,

$$\Pr[A \mid q \leq Q] = \prod_{i=1}^{t} \Pr[A_i \mid q \leq Q] \leq \left(1 - \frac{\varepsilon - 4d\alpha^*}{4d}\right)^t < \frac{1}{10}.$$

It now remains to bound $\Pr[q > Q]$. Let η stand for $1/10t$. Let α_i denote the fraction of erasures in the line sampled during iteration i and let q_i denote the number of queries made by the algorithm during iteration i. Let G denote the (good) event that $\alpha_i \leq \alpha^*/\eta$ for all iterations $i \in [t]$. By Corollary 3.5.4, $\mathbb{E}[q_i \mid G] \leq 5\eta \cdot \log n/(\eta - \alpha^*)$, and by the linearity of expectation,

$$\mathbb{E}[q \mid G] \leq \frac{\log n}{2(\eta - \alpha^*)} \leq \frac{120d \log n}{\varepsilon},$$

where the last inequality above follows from our assumption that $\alpha \leq \varepsilon/248d$. Using Markov's inequality, $\Pr[q > Q \mid G] \leq 1/10$. Also, by combining Lemma 3.6.3 with a union bound over the iterations of Step 2 of the tester, we can see that $\Pr[\overline{G}] \leq 1/10$. Therefore, $\Pr[q > Q] \leq \Pr[q > Q \mid G] + \Pr[\overline{G}] \leq 1/5$. Thus, the probability that the tester does not reject f is

$$\Pr[A] \leq \Pr[A \mid q \leq Q] + \Pr[q > Q] < \frac{1}{10} + \frac{1}{5} < \frac{1}{3}.$$

□

3.7 ERASURE-RESILIENT TESTING OF BOUNDED DERIVATIVE PROPERTIES

In this section, we discuss our erasure-resilient testers for all bounded derivative properties over hypergrid domains and prove Theorem 3.3.2. First, we show in Lemma 3.7.4 that testing for any BDP on $[n]$ reduces to testing monotonicity on $[n]$. Next, we prove Lemma 3.7.7 and Lemma 3.7.8 that reduces the problem of erasure-resilient testing of a BDP over hypergrid domains to testing of the same property over the line.

3.7.1 Erasure-Resilient Bounded Derivative Property Tester for the Line

In this section, we show that erasure-resilient testing of bounded derivative properties (BDPs) on the line reduces to erasure-resilient monotonicity testing on the line and prove Theorem 3.7.5.

Given a function $f : [n] \to \mathbb{R} \cup \{\perp\}$, and a bounded derivative property $\mathcal{P}$, we first define violated pairs in f with respect to $\mathcal{P}$.

Definition 3.7.1 (Violated pair)**.** *Given a function $f : [n] \to \mathbb{R} \cup \{\perp\}$ and bounding family $\mathbf{B}$ consisting of functions $l, u : [n-1] \to \mathbb{R}$, two points $x, y \in \mathcal{N}$ such that $x < y$ violate the property $\mathcal{P}(\mathbf{B})$ with respect to f if $f(x) - f(y) > \mathfrak{m}_{\mathbf{B}}(x, y) = -\sum_{t=x}^{y-1} l(t)$ or $f(y) - f(x) > \mathfrak{m}_{\mathbf{B}}(y, x) = \sum_{t=x}^{y-1} u(t)$. The pairs (x, y) and (y, x) are called violated.*

Consider a bounded derivative property $\mathcal{P}$ of functions defined over $[n]$ and associated bounding functions $l, u : [n-1] \to \mathbb{R}$. The following claim states that, we may assume w.l.o.g. that $l(i) = -u(i)$ for all $i \in [n-1]$. We use it in the proof of Claim 3.7.3.

Claim 3.7.2. *Consider a function $f : [n] \to \mathbb{R} \cup \{\perp\}$ and a bounding function family $\mathbf{B}$ over $[n]$ with $l, u : [n-1] \to \mathbb{R}$. Let $g : [n] \to \mathbb{R} \cup \{\perp\}$ be a function that takes the value $f(i) + \sum_{j=i}^{n-1} \frac{l(j)+u(j)}{2}$ for each $i \in \mathcal{N}$ and is erased on the remaining points. Let $\mathbf{B}'$ be a bounding function family over $[n]$ with $l', u' : [n-1] \to \mathbb{R}$ such that $u'(i) = -l'(i) = \frac{u(i)-l(i)}{2}$ for all $i \in [n-1]$. Then $x, y \in \mathcal{N}$ violate $\mathcal{P}(\mathbf{B})$ with respect to f if and only if x, y violate $\mathcal{P}(\mathbf{B}')$ with respect to g.*

Proof. Note that $x, y \in \mathcal{N}$, where $x < y$, is not violated with respect to f if and only if $\max\{f(x) - f(y) - \mathfrak{m}_{\mathbf{B}}(x,y), f(y) - f(x) - \mathfrak{m}_{\mathbf{B}}(y,x)\} \le 0$. We have

$$\begin{aligned} g(x) - g(y) - \mathfrak{m}_{\mathbf{B}'}(x,y) &= f(x) - f(y) + \sum_{i=x}^{y-1} \frac{u(i)+l(i)}{2} - \sum_{i=x}^{y-1} \frac{u(i)-l(i)}{2} \\ &= f(x) - f(y) - \sum_{i=x}^{y-1} l(i) = f(x) - f(y) - \mathfrak{m}_{\mathbf{B}}(x,y). \end{aligned}$$

Also,

$$\begin{aligned} g(y) - g(x) - \mathfrak{m}_{\mathbf{B}'}(y,x) &= f(y) - f(x) - \sum_{i=x}^{y-1} \frac{u(i)+l(i)}{2} - \sum_{i=x}^{y-1} \frac{u(i)-l(i)}{2} \\ &= f(y) - f(x) - \sum_{i=x}^{y-1} u(i) = f(y) - f(x) - \mathfrak{m}_{\mathbf{B}}(y,x). \end{aligned}$$

Thus, $\max\{g(x) - g(y) - \mathfrak{m}_{\mathbf{B}'}(x,y), g(y) - g(x) - \mathfrak{m}_{\mathbf{B}'}(y,x)\} = \max\{f(x) - f(y) - \mathfrak{m}_{\mathbf{B}}(x,y), f(y) - f(x) - \mathfrak{m}_{\mathbf{B}}(y,x)\}$. The claim follows. $\square$

The following claim shows a reduction from testing BDPs over $[n]$ to testing monotonicity over $[n]$.

Claim 3.7.3. *Consider an α-erased function $f : [n] \to \mathbb{R} \cup \{\perp\}$ and bounding functions $l, u : [n-1] \to \mathbb{R}$ such that $-l(i) = u(i) = \gamma(i)$ for all $i \in [n-1]$. Let $\mathcal{P}$ be the BDP defined by l and u. Let $g, h : [n] \to \mathbb{R} \cup \{\perp\}$ be two functions that take the values $g(i) = f(i) - \sum_{r=i}^{n-1} \gamma(r)$ and $h(i) = -f(i) - \sum_{r=i}^{n-1} \gamma(r)$ for all $i \in \mathcal{N}$ and are erased on the remaining points. Then, the following conditions hold:*

(1) $x, y \in \mathcal{N}$ violate $\mathcal{P}$ with respect to f iff x, y violate monotonicity with respect to either g or h.

(2) If f satisfies $\mathcal{P}$, then both g and h are both monotone.

(3) If f is ε-far from $\mathcal{P}$, then either g or h is at least $\varepsilon/4$-far from monotonicity.

Proof. Consider a pair $(i, j) \in \mathcal{N} \times \mathcal{N}$ where $i < j$. We have,

$$g(i) - g(j) = f(i) - f(j) - \sum_{r=i}^{j-1} \gamma(r);$$
$$h(i) - h(j) = f(j) - f(i) - \sum_{r=i}^{j-1} \gamma(r).$$

If (i, j) does not violate $\mathcal{P}$ with respect to f, we have $f(j) - f(i) - \sum_{r=i}^{j-1} \gamma(r) \leq 0$ and $f(i) - f(j) - \sum_{r=i}^{j-1} \gamma(r) \leq 0$. Thus, (i, j) satisfies the monotonicity property with respect to g and h. On the other hand, if (i, j) violates $\mathcal{P}$ with respect to f, then either $f(j) - f(i) - \sum_{r=i}^{j-1} \gamma(r) > 0$ or $f(i) - f(j) - \sum_{r=i}^{j-1} \gamma(r) > 0$. That is, (i, j) violates monotonicity with respect to either g or h. Parts (1) and (2) of the lemma follow directly from these arguments.

To prove part (3) of the lemma, assume that f is ε-far from the property $\mathcal{P}$. Define the violation graph G_f as follows. The vertex set corresponds to $\mathcal{N}$. For each $(i, j) \in \mathcal{N} \times \mathcal{N}$ such that $i < j$, there is an (undirected) edge between $i \in \mathcal{N}$ and $j \in \mathcal{N}$ iff the pair (i, j) violates $\mathcal{P}$ with respect to f. By (Chakrabarty et al., 2017, Lemma 2.5), the size of every maximal matching in G_f is at least $\varepsilon \cdot n/2$. Consider a maximal matching M in G_f. From the discussion above, the pair of nonerased points corresponding to each edge in M violates monotonicity with respect to either g or h. Therefore, at least $\varepsilon \cdot n/4$ pairs $(i, j) \in \mathcal{N} \times \mathcal{N}$ such that $i < j$, violate monotonicity with respect to at least one of g and h. Assume w.l.o.g. that at least $\varepsilon \cdot n/4$ such pairs violate monotonicity with respect to h. One has to change the function value of h on at least one endpoint of each such pair to repair it. This means that h is at least $\varepsilon/4$-far from monotone. $\square$

Therefore, in order to test the bounded derivative property $\mathcal{P}$ on f with proximity parameter ε, one can test monotonicity on g and h with proximity parameter $\varepsilon/4$ and error probability $1/6$ and accept iff both tests accept.

Lemma 3.7.4. *Let $\alpha \in [0,1), \varepsilon \in (0,1)$ such that $\alpha + \varepsilon < 1$. Let $Q_{\texttt{mon}}(\alpha, \varepsilon, n)$ denote the query complexity of α-erasure-resilient ε-testing of monotonicity of real-valued functions on the line. Then, for every BDP, α-erasure-resilient ε-testing of real-valued functions on the line has query complexity $O(Q_{\texttt{mon}}(\alpha, \varepsilon/4, n))$. The same statement holds for 1-sided error testing.*

The following theorem is a direct consequence of Lemma 3.7.4 and Theorem 3.3.1.

Theorem 3.7.5 (BDP tester on the line)**.** *For every BDP $\mathcal{P}$, there exists a 1-sided error α-erasure-resilient ε-tester for $\mathcal{P}$ of real-valued functions over $[n]$ that works for all $\alpha \in [0,1), \varepsilon \in (0,1)$ such that $\alpha + \varepsilon < 1$, with query complexity $O\left(\frac{\log n}{\varepsilon}\right)$.*

3.7.2 Erasure-Resilient Dimension Reduction

In this section, we prove two important properties of a uniformly random axis parallel line in the hypergrid $[n]^d$. We do this in Lemma 3.7.7 and Lemma 3.7.8, which we jointly call erasure-resilient dimension reduction. We first introduce some notation.

Let g be an α-erased function on $\mathcal{D}$. Recall that the Hamming distance of g from $\mathcal{P}$, denoted by $\text{dist}(g, \mathcal{P})$, is the least number of nonerased points on which every completion of g needs to be changed to satisfy $\mathcal{P}$. The relative Hamming distance between g and $\mathcal{P}$ is $\text{dist}(g, \mathcal{P})/n$. We use $g_{|\mathcal{S}}$ to denote the restriction of g to a subset $\mathcal{S} \subseteq \mathcal{D}$. Note that all these definitions make sense even for functions with no erasures in them.

Let $\mathcal{L}$ denote the set of all *axis-parallel lines* in $[n]^d$. Let $\mathcal{P}$ be a bounded derivative property of functions over $[n]^d$ defined by a bounding family $\mathbf{B} = \{l_1, u_1, \ldots, l_d, u_d\}$

where $l_i, u_i : [n-1] \to \mathbb{R}$ for all $i \in [d]$. For $i \in [d]$, let $\mathcal{P}^i$ denote the set of functions over $[n]^d$ with no violations to $\mathcal{P}$ along dimension i. Let $\mathcal{P}^i_{\mathsf{line}}$ denote the bounded derivative property of functions over $[n]$ defined by the bounding functions $l_i, u_i : [n-1] \to \mathbb{R}$.

Consider an α-erased function $f : [n]^d \to \mathbb{R} \cup \{\perp\}$. Let $\mathcal{N} \subseteq [n]^d$ denote the set of nonerased points in f. For an axis-parallel line $\ell \in \mathcal{L}$, let $\mathcal{N}_\ell$ denote the set of nonerased points on ℓ and f_ℓ denote the function f restricted to ℓ.

Lemma 3.7.7 shows that, for a uniformly random axis-parallel line $\ell \in \mathcal{L}$, the expected relative Hamming distance of f_ℓ from $\mathcal{P}^i_{\mathsf{line}}$ is roughly proportional to the relative Hamming distance of f from $\mathcal{P}$, where i is the dimension along which ℓ lies. First, we prove Claim 3.7.6 that we use in our proof of Lemma 3.7.7.

Claim 3.7.6. *Let $\alpha \in [0,1)$. For every α-erased function $f : [n]^d \to \mathbb{R} \cup \{\perp\}$ and every bounded derivative property $\mathcal{P}$ over $[n]^d$, we have,*

$$\frac{1}{4} dist(f, \mathcal{P}) \le \alpha \cdot d \cdot n^d + \sum_{i=1}^{d} dist(f, \mathcal{P}^i).$$

Proof. Let $g : [n]^d \to \mathbb{R}$ be a function in $\mathcal{P}$ such that $\mathrm{dist}(g_{|\mathcal{N}}, f_{|\mathcal{N}})$ is minimum. We define $f_* : [n]^d \to \mathbb{R}$, a completion of f, such that $f_*(x) = f(x)$ for all $x \in \mathcal{N}$ and $f_*(x) = g(x)$ for all $x \notin \mathcal{N}$.

For all $i \in [d]$, let $g^i : [n]^d \to \mathbb{R}$ in $\mathcal{P}^i$ be such that $\mathrm{dist}(g^i_{|\mathcal{N}}, f_{|\mathcal{N}})$ is minimum. Also, for all $i \in [d]$, let $h^i : [n]^d \to \mathbb{R}$ be defined as $h^i(x) = f(x)$ for all $x \in \mathcal{N}$, and $h^i(x) = g^i(x)$ for $x \notin \mathcal{N}$. Note that for all $i \in [d]$, the function h^i is a completion of f. We have,

$$\frac{1}{4}\mathrm{dist}(f, \mathcal{P}) \le \frac{1}{4}\mathrm{dist}(f_*, \mathcal{P}) \quad (\because \text{ as } f_* \text{ is a completion of } f)$$

$$\leq \sum_{i=1}^{d} \operatorname{dist}(f_*, \mathcal{P}^i) \quad (\because \text{ dimension reduction (Chakrabarty et al. (2017))})$$

$$\leq \sum_{i=1}^{d} \operatorname{dist}(f_*, g^i) \quad (\because g^i \in \mathcal{P}^i)$$

$$\leq \sum_{i=1}^{d} \operatorname{dist}(f_*, h^i) + \sum_{i=1}^{d} \operatorname{dist}(h^i, g^i) \qquad \text{(triangle inequality)}$$

$$\leq d \cdot \alpha \cdot n^d + \sum_{i=1}^{d} \operatorname{dist}(f, \mathcal{P}^i).$$

To see the last inequality, notice that f_* and h^i differ only on points in $[n]^d \setminus \mathcal{N}$. Hence, for all $i \in [d]$, we have, $\operatorname{dist}(f_*, h^i) \leq \alpha \cdot n^d$. Also, for all $i \in [d]$, $\operatorname{dist}(f, \mathcal{P}^i)$ is defined as the minimum number of points in $\mathcal{N}$ that every completion of f need to be changed to get a function in $\mathcal{P}^i$. For $i \in [d]$, since the function h^i is a completion of f and g^i is the function that minimizes $\operatorname{dist}(g^i_{|\mathcal{N}}, f_{|\mathcal{N}})$, we can see that $\operatorname{dist}(f, \mathcal{P}^i) \geq \operatorname{dist}(h^i, g^i)$. □

We now use Claim 3.7.6 to prove the first part of our dimension reduction.

Lemma 3.7.7 (Dimension reduction: distance). *Let ε_f be the relative Hamming distance of f from $\mathcal{P}$. Given $\ell \in \mathcal{L}$, let ε_ℓ denote the relative Hamming distance of f_ℓ from $\mathcal{P}^i_{\mathsf{line}}$, where $i \in [d]$ is the dimension along which ℓ lies. Then*

$$\mathbb{E}_{\ell \sim \mathcal{L}}[\varepsilon_\ell] \geq \frac{\varepsilon_f}{4d} - \alpha.$$

Proof. There are d axis-parallel directions and, therefore, dn^{d-1} axis-parallel lines in $[n]^d$. Thus, $\Pr[E_\ell] = 1/dn^{d-1}$, where E_ℓ is the event of getting a specific axis parallel line ℓ while sampling u.a.r. from $\mathcal{L}$. Let $\mathcal{L}_i$ denote the set of axis parallel lines along dimension i.

$$\begin{aligned}
\mathbb{E}_{\ell\sim\mathcal{L}}[\varepsilon_\ell] &= \sum_{\ell\in\mathcal{L}} \varepsilon_\ell \cdot \Pr[E_\ell] \\
&= \sum_{i=1}^{d}\sum_{\ell\in\mathcal{L}_i} \varepsilon_\ell \cdot \Pr[E_\ell] \\
&= \frac{1}{dn^{d-1}} \cdot \sum_{i=1}^{d}\sum_{\ell\in\mathcal{L}_i} \frac{\operatorname{dist}(f_\ell, \mathcal{P}^i_{\mathsf{line}})}{n} \\
&= \frac{1}{dn^{d}} \cdot \sum_{i=1}^{d}\sum_{\ell\in\mathcal{L}_i} \operatorname{dist}(f_\ell, \mathcal{P}^i_{\mathsf{line}}) \\
&= \frac{1}{dn^{d}} \cdot \sum_{i=1}^{d} \operatorname{dist}(f, \mathcal{P}^i) \\
&\geq \frac{1}{dn^{d}} \cdot \left(\frac{\operatorname{dist}(f,\mathcal{P})}{4} - \alpha d \cdot n^d\right) && \text{by Claim 3.7.6} \\
&\geq \frac{\varepsilon_f}{4d} - \alpha.
\end{aligned}$$

□

We conclude this section with the second part of our dimension reduction.

Lemma 3.7.8 (Dimension reduction: fraction of erasures)**.** *Consider an α-erased function $f : [n]^d \to \mathbb{R} \cup \{\perp\}$. Given an axis-parallel line $\ell \in \mathcal{L}$, let α_ℓ denote the fraction of erased points in ℓ. Then, for every $\eta \in (0,1)$,*

$$\Pr_{\ell\sim\mathcal{L}}[\alpha_\ell > \alpha/\eta] \leq \eta.$$

Proof. Note that a uniformly randomly sampled point in $[n]^d$ is erased with probability α. We can sample a point uniformly at random by first sampling a line $\ell \in \mathcal{L}$ uniformly at random and then sampling a point uniformly randomly on ℓ, which is erased with probability α_ℓ. Therefore we have

$$\alpha = \sum_{\ell \in \mathcal{L}} \Pr[E_\ell] \cdot \alpha_\ell = \mathbb{E}_{\ell \sim \mathcal{L}}[\alpha_\ell].$$

The claim then follows from Markov's inequality. □

3.7.3 Erasure-Resilient Bounded Derivative Property Testers for the Hypergrids

We now present our erasure-resilient tester for an arbitrary BDP $\mathcal{P}$ and complete the proof of Theorem 3.3.2. Let $\mathbf{B} = \{\ell_i, u_i : i \in [d]\}$ be a bounding family for $\mathcal{P}$, where $\ell_i, u_i : [n-1] \to \mathbb{R}$. Let $\mathcal{L}_i$ denote the set of axis-parallel lines along dimension i. Our tester is described in Algorithm 3.4.

Algorithm 3.4 Erasure-Resilient Tester for BDP $\mathcal{P}$ over $[n]^d$

```
Input: parameters $\varepsilon \in (0,1), \alpha \in [0, \varepsilon/968d]$; oracle access to $f : [n]^d \to \mathbb{R} \cup \{\perp\}$
1: Set $Q = \left\lceil \frac{4800d \cdot \log n}{\varepsilon} \right\rceil$.
2: repeat $\frac{48d}{\varepsilon - 4d\alpha}$ times:
3:     Sample a line $\ell \in \mathcal{L}$ uniformly at random.
4:     Define $g$ and $h$ from $f_\ell$, $\ell_i$ and $u_i$ as in Claim 3.7.3 if $\ell$ is sampled from $\mathcal{L}_i$.
5:     Sample points u.a.r. from $\ell$ and query them until we get a point $s \in \mathcal{N}$.
6:     Perform a randomized binary search for $s$ on $\ell$ as in Algorithm 3.2.
7:     Reject if any violation to monotonicity is found in either $g$ or $h$.
8: Accept at any point if the number of queries exceed $Q$.
```

The bound on the query complexity of the tester is evident from its description. We will now prove its correctness in Lemma 3.7.9, which will then imply Theorem 3.3.2. The proof of Lemma 3.7.9 is very similar to the proof of Lemma 3.6.4. The only difference is that we use an additional step in the analysis to reduce BDP testing to monotonicity testing over the line, given by Claim 3.7.3. This step introduces constant-factor differences in the mathematical expressions.

Lemma 3.7.9. *Algorithm 3.4 accepts if f is in $\mathcal{P}$, and rejects with probability at least $2/3$ if f is ε-far from $\mathcal{P}$.*

Proof. To prove the first part of the lemma, consider an α-erased function $f \in \mathcal{P}$ and consider an arbitrary iteration of the tester. Suppose the tester samples a line $\ell \in \mathcal{L}$ such that ℓ is along the i^{th} dimension. Let $g_\ell, h_\ell : [n] \to \mathbb{R} \cup \{\perp\}$ denote the functions g and h obtained by applying Claim 3.7.3 to f_ℓ and $\mathcal{P}^i_{\mathsf{line}}$. By Claim 3.7.3, we know that f_ℓ is in $\mathcal{P}^i_{\mathsf{line}}$ iff both h_ℓ and g_ℓ are monotone. As is clear from Algorithm 3.4, the tester runs (1-sided error) erasure-resilient monotonicity testers for two such functions and therefore, the tester accepts f in that iteration. Hence, the tester accepts f.

Assume that f is ε-far from $\mathcal{P}$. Let $\alpha^* \le \alpha$ be the actual fraction of erasures in f. As before, we analyze a 'thought tester' that is identical to Algorithm 3.4 except that it runs Step 2 only $\frac{48d}{\varepsilon - 4d\alpha^*}$ times.

Let A denote the event that the tester does not reject f in any of its iterations. If q denotes the total number of queries made by the tester, we have, $\Pr[A] \le \Pr[A \mid q \le Q] + \Pr[q > Q]$.

Let t denote $\frac{48d}{\varepsilon - 4d\alpha^*}$, the number of iterations of the tester. Let A_i denote the event that the tester accepts in its i^{th} iteration. As before, let E_ℓ denote the event that the tester gets the line ℓ when it samples lines u.a.r. from $\mathcal{L}$. Let ε_ℓ denote the relative Hamming distance of f_ℓ from $\mathcal{P}^j_{\mathsf{line}}$, where j is the index of the dimension along which ℓ lies. By Claim 3.7.3, either g_ℓ or h_ℓ is $\varepsilon_\ell/4$-far from monotone. Thus, the tester rejects with probability at least $\varepsilon_\ell/4$ if it samples ℓ. We then have,

$$\Pr[A_i \mid q \le Q] \le \sum_{\ell \in \mathcal{L}} \left(1 - \frac{\varepsilon_\ell}{4}\right) \Pr[E_\ell] = 1 - \frac{1}{4} \cdot \mathbb{E}_{\ell \sim \mathcal{L}}[\varepsilon_\ell].$$

Using Lemma 3.7.7 and the fact that $\varepsilon_f \geq \varepsilon$, we have,

$$\mathbb{E}_{\ell\sim\mathcal{L}}[\varepsilon_\ell] \geq \frac{\varepsilon_f}{4d} - \alpha^* \geq \frac{\varepsilon}{4d} - \alpha^*.$$

Therefore,

$$\Pr[A \mid q \leq Q] = \prod_{i=1}^{t} \Pr[A_i \mid q \leq Q] \leq \left(1 - \frac{\varepsilon - 4d\alpha^*}{16d}\right)^t < \frac{1}{10}.$$

It now remains to bound $\Pr[q > Q]$. Let η stand for $1/10t$. Let α_i denote the fraction of erasures in the line sampled during iteration i and let q_i denote the number of queries made by the algorithm during iteration i. Let G denote the (good) event that $\alpha_i \leq \alpha^*/\eta$ for all iterations $i \in [t]$. By Corollary 3.5.4, $\mathbb{E}[q_i \mid G] \leq 5\eta \cdot \log n/(\eta - \alpha^*)$, and by the linearity of expectation,

$$\mathbb{E}[q \mid G] \leq \frac{\log n}{2(\eta - \alpha^*)} \leq \frac{480d \log n}{\varepsilon},$$

where the last inequality follows from our assumption that $\alpha \leq \varepsilon/968d$. Using Markov's inequality,

$$\Pr[q > Q \mid G] \leq \frac{1}{10}.$$

Also, by combining Lemma 3.7.8 with a union bound, we can see that $\Pr[\overline{G}] \leq 1/10$. Therefore,

$$\Pr[q > Q] \leq \Pr[q > Q \mid G] + \Pr[\overline{G}] \leq \frac{1}{5}.$$

□

3.7.4 Limitations of Dimension Reduction in Erasure-Resilient Testing

In this section, we show that when the fraction of erasures is large enough, dimension reduction based testers that sample axis parallel lines uniformly at random and check for violations on them, are bound to fail. Axis-parallel lines in hypercubes (Definition 3.2.1) are called *edges*. We prove the following claim.

Lemma 3.7.10. *For all $\varepsilon \in (1/e^2, 1/2)$, all large enough even d and $\alpha = \Theta(\varepsilon/\sqrt{d})$, there exists an α-erased function $f : \{0,1\}^d \to \mathbb{R} \cup \{\perp\}$, such that f is ε-far from monotone but f has no violations to monotonicity along the edges of the hypercube $\{0,1\}^d$.*

Proof. Let $\varepsilon \in (\frac{1}{e^2}, 1/2)$. Fix a large enough even number $d \in \mathbb{N}$. Let $k \in (d/2 - \sqrt{d}, d/2)$ be the smallest natural number such that $\varepsilon \cdot 2^d \leq \sum_{i=0}^{k} \binom{d}{i}$. Such a k has to exist because $\sum_{i=0}^{d/2} \binom{d}{i} = \frac{1}{2} \cdot 2^d$ and $\sum_{i=0}^{d/2-\sqrt{d}} \binom{d}{i} < \frac{1}{e^2} \cdot 2^d$, where the latter inequality follows by an application of the Hoeffding bound to an appropriately defined binomial random variable.

For $x \in \{0,1\}^d$, let $||x||_0$ denote the number of nonzero coordinates in x. The function f, for $x \in \{0,1\}^d$ is defined as:

$$f(x) = \begin{cases} \perp & \text{if } ||x||_0 = k+1 \\ 1 & \text{if } ||x||_0 > k+1 \\ 0 & \text{otherwise.} \end{cases}$$

The fraction of erasures α^* in f is equal to $\frac{\binom{d}{k+1}}{2^d}$. Recall that the distance to monotonicity of a function $f : \{0,1\}^d \to \{0,1\}$, denoted by ε_f, is the fraction of nonerased function values that needs to be changed in every completion of f to make it monotone. We can see that $\varepsilon_f = \frac{\sum_{i=0}^{k} \binom{d}{i}}{2^d}$, since we need to change all the function values "below the erased layer of the hypercube" in order to make f monotone. We can also see that no edge of the hypercube is violated with respect to monotonicity.

Since $\frac{d}{2} - (k+1) = o(n^{2/3})$, we can use the fact shown by Spencer & Florescu (2014) that $\frac{\binom{d}{k+1}}{2^d} = \Theta\left(\frac{1}{\sqrt{d}}\right)$. Putting everything together, we get, $\alpha^*/\varepsilon_f = \Theta(1/\sqrt{d})$. This completes the proof of the lemma. □

3.8 ERASURE-RESILIENT CONVEXITY TESTER FOR THE LINE

In this section, we prove Theorem 3.3.3. Given an α-erased function $f : [n] \to \mathbb{R} \cup \{\perp\}$, let ν_i denote the i-th nonerased domain point in $[n]$. The derivative of f at a point $\nu_i \in \mathcal{N}$, denoted by $\Delta f(\nu_i)$, is $\frac{f(\nu_{i+1}) - f(\nu_i)}{\nu_{i+1} - \nu_i}$, whenever $\nu_{i+1} \leq n$. The function f is convex iff $\Delta f(\nu_i) \leq \Delta f(\nu_{i+1})$ for all $i \in [|\mathcal{N}| - 2]$. In other words, a function is convex iff its derivative is monotone.

Looking at the above definition, it would seem that testing convexity of a function can be reduced to testing monotonicity of its derivative. However, this reduction does not work even for the case of testing when there are no erasures, since there are functions that are very far from convex but whose derivatives are very close to monotone. One such example, given by Parnas et al. (2003), is the following function $f : [n] \to \mathbb{R}$ defined for even n. For $i \leq n/2$, we have $f(i) = i$ and for $i > n/2$, we have $f(i) = i - 1$. This function is $\frac{1}{2}$-far from convex. But its derivative, Δf, takes the value 1 on all points except for a single point, where it is 0. Hence, Δf is $\frac{1}{n-1}$-close to monotone.

Parnas et al. (2003) then describe a convexity tester by utilizing the relationship between convexity of a function and the monotonicity of its derivative more cleverly. Our tester builds upon the ideas of the convexity tester of Parnas et al. (2003).

A high-level idea of the tester is as follows. Our tester (Algorithm 3.5) has several iterations. Every iteration of the tester can be thought of as a traversal of a uniformly random rooted path in a random binary search tree $\mathcal{T}$ on $\mathcal{N}$ sampled from the distribution $\mathcal{D}_{\mathcal{T}}$, just as Algorithm 3.2. For each interval on such a path, we check a set of conditions computed based on the values at some nonerased points in the interval, called *anchor points*, and two real numbers, called the left and right slopes. More specifically, we verify that the function restricted to the sampled nonerased points in the interval is convex, by comparing the slopes across consecutive points. The algorithm accepts if all the intervals it sees pass these checks. The main steps

Algorithm 3.5 Erasure-Resilient Convexity Tester

Input: parameters $\varepsilon \in (0,1), \alpha \in [0,1)$; oracle access to $f : [n] \to \mathbb{R} \cup \{\perp\}$.

```
1: Set $Q = \lceil \frac{180 \log n}{\varepsilon} \rceil$.
2: **Accept** at any point if the number of queries exceeds $Q$.
3: **repeat** $\frac{2}{\varepsilon}$ times
4:     Sample points in $I[1,n]$ u.a.r and query them until we get a point $s \in \mathcal{N}$.
5:     TEST-INTERVAL($I[1,n], \emptyset, -\infty, +\infty, s$) and **Reject** if it rejects.
6: **Accept**.
```

in the analysis of the tester follow that of the analysis of Algorithm 3.2. To analyze the tester, we first prove that, with high probability, the algorithm does not run out of its budget of queries Q. For this, we classify the queries that the tester makes into two kinds as follows and analyze them separately.

Definition 3.8.1 (Sampling Queries)**.** *The queries made by the tester when it repeatedly samples and queries points from an interval until it finds a nonerased domain point are called sampling queries.*

Definition 3.8.2 (Walking Queries)**.** *The queries made by the tester when it keeps querying consecutive points from an interval, starting from one nonerased point until it finds the next nonerased point, are called walking queries.*

In the proof of Lemma 3.8.3, we first show that the expected number of walking queries is at most twice the number of the expected number of the sampling queries and then use Corollary 3.5.4 to bound the expected number of sampling queries.

In the second part of the analysis we prove that, conditioned on the total number of queries made by the algorithm not exceeding Q, in each iteration, with probability at least ε, the tester rejects a function that is ε-far from convex. This part of the proof draws ideas from the proof of correctness of the tester in Parnas et al. (2003).

Procedure 3.6 TEST-INTERVAL$(I[i,j], \mathcal{A}, m_\ell, m_r, s)$

Input: interval $I[i,j]$; a set of nonerased points $\mathcal{A}$; left slope $m_\ell \in \mathbb{R}$; right slope $m_r \in \mathbb{R}$; search point $s \in \mathcal{N}$.

```
1:  Sample points u.a.r. from $I[i,j]$ and query them until we get a point $x \in \mathcal{N}$.
2:  Sequentially query points $x+1, x+2 \dots$ until we get a nonerased point $y$.
3:                  ▷ Set $y \leftarrow x$ if there is no nonerased point in $I[i,j]$ to the right of $x$.
4:  Sequentially query points $x-1, x-2 \dots$ until we get the nonerased point $z$.
5:                  ▷ Set $z \leftarrow x$ if there is no nonerased point in $I[i,j]$ to the left of $x$.
6:  Let $(a_1, a_2, \dots, a_k)$ denote the sorted list of points in the set $\mathcal{A}' \leftarrow \mathcal{A} \cup \{x, y, z\}$.
7:  Let $m_i = (f(a_{i+1}) - f(a_i))/(a_{i+1} - a_i)$ for all $i \in [k-1]$.
8:  Reject if $m_\ell \le m_1 \le m_2 \le \dots \le m_{k-1} \le m_r$ is not true.
9:  Let $\mathcal{A}'_\ell$ and $\mathcal{A}'_r$ be the sets of points in $\mathcal{A}'$ that are smaller and larger than $x$,
    respectively.
10: if $s < x$ then
11:     Reject if TEST-INTERVAL$(I[i,z], \mathcal{A}'_\ell, m_\ell, \Delta f(z), s)$ rejects.
12: if $s > x$ then
13:     Reject if TEST-INTERVAL$(I[y,j], \mathcal{A}'_r, \Delta f(x), m_r, s)$ rejects.
14: Accept.
```

Lemma 3.8.3. *Algorithm 3.5 accepts if f is convex, and rejects with probability at least $2/3$ if f is ε-far from convex.*

Proof. We first define some notation for our analysis. Consider a search path traversed by the algorithm. Similar to the analysis of Algorithm 3.2, this path can be viewed as a uniformly random rooted path in a binary tree $\mathcal{T}$ over $\mathcal{N}$, sampled according to $\mathcal{D}_\mathcal{T}$. Let $I[i,j]$ be an interval on the path. Consider the execution of TEST-INTERVAL

(Procedure 3.6) called with $I[i, j]$ as the first argument. We call the nonerased point x sampled in Step 1 of Procedure 3.6 its *pivot*, the set of points $\mathcal{A}'$ in Step 6 of Procedure 3.6 its *anchor set* and the values m_ℓ and m_r (passed to the procedure Test-Interval) as its *left* and *right slopes*, respectively. That is, given a binary search tree $\mathcal{T}$ over $\mathcal{N}$, we associate each node in the tree with an interval, a pivot, an anchor set and two slopes. Note that the size of the anchor set $\mathcal{A}'$ in Step 6 is at most 5, since each interval can have at most two anchor points of its ancestors carried down to it (the extreme nonerased points of the interval), and also have at most 3 of its own anchor points.

It is evident that the tester accepts whenever f is convex. To prove the other part of the lemma, assume that f is ε-far from convex. Let $\alpha^* \leq \alpha$ denote the actual fraction of erasures in f. As before, it is enough to analyze a "thought tester" that is identical to Algorithm 3.5 in all respects except that it runs Step 3 only $2(1 - \alpha^*)/\varepsilon$ times.

Let A be the event that the tester accepts f. Let q denote the total number of queries made. We have, $\Pr[A] \leq \Pr[A \mid q \leq Q] + \Pr[q > Q]$.

We first bound $\Pr[q > Q]$. As mentioned earlier, the queries made by the tester can be classified into sampling queries (Definition 3.8.1) and walking queries (Definition 3.8.2). By Corollary 3.5.4, the expected number of sampling queries made in one iteration of the tester is at most $5 \log n/(1 - \alpha^*)$.

We will now bound the expected number of walking queries. Consider an interval I with α_I fraction of erasures in it. A point in I can get queried as part of the walking queries if either the first nonerased point to its right or the first nonerased point to its left on the line $[n]$ gets sampled as the pivot of I. For a nonerased point $i \in I$, let $w(i)$ denote the number of walking queries to be made if the algorithm samples i as the pivot. Therefore

$$\sum_{i \in \mathcal{N} \cap I} w(i) \leq 2|I|,$$

since every point in I gets counted at most twice in this sum. There are exactly $|I|(1-\alpha_I)$ nonerased points in I and each of them could be the pivot in I with equal probability. Hence, the expected number of walking queries that Algorithm 3.5 makes in I is at most $2/(1-\alpha_I)$. This is at most twice the expected number of sampling queries that the algorithm makes in I.

Therefore, by the linearity of expectation, the expected number of walking queries made in one iteration of the tester is at most $10 \log n/(1-\alpha^*)$. Thus, the expected value of the total number of queries made by the tester in one iteration is at most $15 \log n/(1-\alpha^*)$ and that over all iterations is at most $30(1-\alpha^*) \log n/\varepsilon(1-\alpha^*) = 30 \log n/\varepsilon$. Thus, by Markov's inequality,

$$\Pr[q > Q] \leq \frac{1}{6}.$$

Next, we bound $\Pr[A|q \leq Q]$. Consider a binary search tree $\mathcal{T}$ over $\mathcal{N}$, sampled according to the distribution $\mathcal{D}_{\mathcal{T}}$, and a function $f : [n] \to \mathbb{R} \cup \{\perp\}$. Let $\Gamma(I[i,j], \mathcal{A}, m_\ell, m_r)$ be a node in $\mathcal{T}$ with interval $I[i,j]$, anchor set $\mathcal{A} = \{a_1, \ldots, a_k\} \subseteq I[i,j]$ and slopes m_ℓ and m_r such that $a_i \leq a_{i+1}$ for all $i \in [k-1]$. Let $m_i = (f(a_{i+1}) - f(a_i))/(a_{i+1} - a_i)$ for all $i \in [k-1]$.

Definition 3.8.4 (Good Node, Bad Node)**.** *A node $\Gamma(I[i,j], \mathcal{A}, m_\ell, m_r)$ is good if $m_\ell \leq m_1 \leq m_2 \leq \cdots \leq m_{k-1} \leq m_r$. Otherwise, it is bad.*

Definition 3.8.5 (Violator Node)**.** *An node Γ is a violator if it is bad and all its ancestor nodes in $\mathcal{T}$ are good.*

Definition 3.8.6 (Witness)**.** *A nonerased domain point is a witness with respect to*

$\mathcal{T}$ if it belongs to an interval associated with a violator node in $\mathcal{T}$.

We prove that if f is ε-far from convex, then, for every binary search tree $\mathcal{T}$, the fraction of nonerased domain points that are witnesses is at least $\frac{\varepsilon}{1-\alpha^*}$. We start by assuming that there is a tree in which the fraction of witnesses is less than $\frac{\varepsilon}{1-\alpha^*}$. We show that we can correct the function values only on the witnesses and get a convex function, which gives a contradiction.

Claim 3.8.7. *If f is ε-far from convex, then the fraction of witnesses in every binary search tree $\mathcal{T}$ is at least $\frac{\varepsilon}{1-\alpha^*}$.*

Proof. Assume for the sake of contradiction that there is a binary search tree $\mathcal{T}$ such that the fraction of witnesses with respect to $\mathcal{T}$ is less than $\frac{\varepsilon}{1-\alpha^*}$. In the following, we will construct a convex function $g : [n] \to \mathbb{R} \cup \{\perp\}$ by changing the values of f only on witnesses with respect to $\mathcal{T}$. Since the fraction of witnesses is less than $\frac{\varepsilon}{1-\alpha^*}$, functions f and g will differ on less than an $\frac{\varepsilon n}{1-\alpha^*}$ fraction of nonerased domain points, which will give us the desired contradiction.

Consider a violator node Γ in $\mathcal{T}$ and let $I[i,j]$ be the interval associated with it. If Γ is the root of $\mathcal{T}$, every nonerased domain point in f is a witness by definition. This contradicts our assumption that the fraction of witnesses is less than $\frac{\varepsilon}{1-\alpha^*}$. Therefore, we can assume that Γ is a non-root node in $\mathcal{T}$. Let the anchor set and slopes associated with the *parent node* of Γ in $\mathcal{T}$ be $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$, m_ℓ and m_r, respectively. Assume without loss of generality that $a_i \leq a_{i+1}$ for all $i \in [k-1]$. Suppose that Γ is the right child of its parent. The case when Γ is the left child of its parent is analogous and is hence omitted. Let $\{a_u, a_{u+1}, \ldots, a_k\}$ be the set of points common to $I[i,j]$ and $\mathcal{A}$. By definition, a_u is the smallest nonerased domain point in $I[i,j]$. Also, the left slope of $I[i,j]$ is $(f(a_u) - f(a_{u-1}))/(a_u - a_{u-1})$ and its right slope is equal to m_r.

Let $m_v = (f(a_{v+1}) - f(a_v))/(a_{v+1} - a_v)$ for all integers v such that $v \in [u-1, k)$. We define g as follows.

- For each $t \in \{a_u, a_{u+1}, \ldots, a_k\}$, set $g(t) = f(t)$.
- For each integer $v \in [u, k)$ and $t \in \mathcal{N} \cap (a_v, a_{v+1})$, set
$$g(t) = f(a_v) + m_v \cdot (t - a_v)$$
- For each $t \in \mathcal{N}$ such that $j \geq t > a_k$, set
$$g(t) = f(a_k) + m_{k-1} \cdot (t - a_k).$$

Since Γ is a violator node, the parent node of Γ is good, by definition. This implies that $m_{u-1} \leq m_u \leq \ldots \leq m_{k-1} \leq m_r$. Therefore, the derivatives of nonerased points in $I[i, j]$ are non-decreasing with respect to g, by virtue of our assignment.

To prove that g is convex, we first show that every node in $\mathcal{T}$ is good with respect to g.

1. Consider a node Γ in $\mathcal{T}$ that is good with respect to f. Let $I[i, j]$ be the interval associated with Γ. If Γ has no ancestors or descendants that are violators (w.r.t. f), it remains good with respect to g as well, since $g(t) = f(t)$ for all $t \in I[i, j]$.
2. Consider a node Γ in $\mathcal{T}$ such that Γ and its ancestors are all good w.r.t. f. Let I be the interval associated with Γ. To prove that Γ is good w.r.t. g, it is enough to show that $f(t) = g(t)$ for every anchor point $t \in I$ of Γ. Note that the only points $t \in I$ for which $f(t)$ and $g(t)$ could be different are the points belonging to intervals associated with violator nodes in $\mathcal{T}$ that are descendants of Γ. Consider a node Γ' in $\mathcal{T}$ such that (1) Γ' is a descendant of Γ, and (2) Γ' is

a violator node w.r.t. f. Let I' be the interval associated with Γ'. The definition of g on points in I' ensures that $g(t) = f(t)$ for every point t common to the anchor sequence of Γ and the interval I'. Thus, we can see that $f(t) = g(t)$ for every anchor point $t \in I$ of Γ. Hence, Γ remains good with respect to g.

3. Consider a node Γ that is either a violator node or has a violator ancestor Γ (w.r.t. f). Let I and I' be the intervals associated with Γ and Γ' respectively. By definition, the parent of Γ' is good with respect to f. Therefore, by the definition of g on I', we have $\Delta g(t-1) \leq \Delta g(t)$ for all $t \in \mathcal{N} \cap I'$. Therefore, Γ' is good with respect to g, and hence Γ is also good with respect to g.

We proved that every node in the tree $\mathcal{T}$ is good with respect to g. We now prove that g is convex. Consider a point $\nu_t \in \mathcal{N}$ such that $2 \leq t \leq |\mathcal{N}| - 1$, where ν_i denotes the i^{th} nonerased point in $[n]$. This point occurs in $\mathcal{T}$ either as the pivot of a non-leaf node or as the sole nonerased domain point in the interval associated with a leaf node. In the former case, the condition $\Delta g(\nu_{t-1}) \leq \Delta g(\nu_t)$ is part of the *goodness condition* of the corresponding node and is satisfied. In the latter case, $\Delta g(\nu_{t-1})$ and $\Delta g(\nu_t)$ are the left and right slopes of the leaf and are compared as part of the goodness condition of the leaf. Thus, $\Delta g(\nu_{t-1}) \leq \Delta g(\nu_t)$ for all $\nu_t \in \mathcal{N}$ such that $2 \leq t \leq |\mathcal{N}| - 1$. Thus, g is convex. □

We conclude our analysis by bounding the probability that the tester does not find a violation. Since the search point s is chosen uniformly at random from the set of nonerased domain points, the probability that it is a witness is at least $\frac{\varepsilon}{1-\alpha^*}$ and thus, the tester detects a violation to convexity with probability at least $\frac{\varepsilon}{1-\alpha^*}$ in every iteration. Therefore,

$$\Pr[A \mid q \leq Q] \leq \left(1 - \frac{\varepsilon}{1-\alpha^*}\right)^{\frac{2(1-\alpha^*)}{\varepsilon}} < \frac{1}{6}.$$

□

3.9 CONNECTION BETWEEN STANDARD, ERASURE-RESILIENT, AND TOLERANT TESTING

In this section, we show that tolerant testing implies erasure-resilient testing, which is in turn implies standard testing. We first show that an erasure-resilient tester for a property is also a standard tester for the same property. We then show that the existence of a fully tolerant tester for a property implies the existence of an erasure-resilient tester for that property, and prove Observation 3.9.2. We also state and prove a slightly different version of Observation 3.9.2 for distance approximation algorithms, and apply that latter version to design erasure-resilient testers for sortedness, monotonicity, and convexity.

Observation 3.9.1 formalizes the intuition that an erasure-resilient tester for a property of functions also serves as a standard tester for the same property.

Observation 3.9.1. *Let $Q(\cdot,\cdot,\cdot)$ be a function that is nondecreasing in its first input. Let $\alpha \in [0,1), \varepsilon \in (0,1)$ be such that $\alpha + \varepsilon < 1$. If A is an α-erasure-resilient ε-tester with query complexity $Q(\alpha, \varepsilon, |\mathcal{D}|)$ for a property $\mathcal{P}$ of functions of the form $f : \mathcal{D} \to \mathcal{R}$, then A is also an ε-tester for $\mathcal{P}$ with query complexity $Q(0, \varepsilon, |\mathcal{D}|)$.*

Proof. As mentioned in Remark 3.1.3, it is natural to assume that an α-erasure-resilient ε-tester is also an α'-erasure-resilient ε-tester for all $\alpha' \leq \alpha$. Therefore, A is a 0-erasure-resilient ε-tester for $\mathcal{P}$ with query complexity $Q(0, \varepsilon, |\mathcal{D}|)$. Since a 0-erasure-resilient ε-tester is an ε-tester, the theorem follows. □

Observation 3.9.2. *Let $\varepsilon_1, \varepsilon_2 \in (0,1)$ be such that $\varepsilon_1 < \varepsilon_2$ and $\varepsilon_1 + \varepsilon_2 < 1$. If A is an $(\varepsilon_1, \varepsilon_2)$-tolerant tester for a property $\mathcal{P}$ of functions of the form $f : \mathcal{D} \to \mathcal{R}$, then* *there exists an ε_1-erasure-resilient ε_2-tester for $\mathcal{P}$ that has the same query complexity as A.*

Proof. Let e be an arbitrary element in the range $\mathcal{R}$. For an ε_1-erased function f, let f_e^r denote the completion of f obtained by assigning the value e to all the erased points.

An ε_1-erasure-resilient ε_2-tester A' for $\mathcal{P}$, when given oracle access to an ε_1-erased function $f : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$, simulates oracle access to the the function f_e^r, runs algorithm A on f_e^r and accepts if and only if A accepts.

A and A' have the same query complexity.

If f satisfies $\mathcal{P}$ there is a completion f' of f that satisfies $\mathcal{P}$ and every other completion of f is ε_1-close to f'. Therefore, f_e^r is ε_1-close to $\mathcal{P}$. Since A accepts with probability at least $2/3$, every function that is ε_1-close to $\mathcal{P}$, the tester A' also accepts f with probability at least $2/3$.

If f is ε_2-far from $\mathcal{P}$, then every completion of f is ε_2-far from $\mathcal{P}$. This is, in particular, true for f_e^r. Since A rejects with probability at least $2/3$, every function that is ε_2-far from $\mathcal{P}$, the tester A' rejects f with probability at least $2/3$. □

Tolerant testers are intimately connected to algorithms that approximate the distance of a function to a specified property, when given oracle access to the function. For a property $\mathcal{P}$ and a function f, we denote by $\varepsilon_{\mathcal{P}}(f)$ the relative Hamming distance of f to $\mathcal{P}$.

Definition 3.9.3 (Distance Approximation Algorithm)**.** *Let $\eta \geq 1$ and $\delta \in [0, 1)$. An algorithm A is a η-multiplicative δ-additive distance approximation algorithm for $\mathcal{P}$, if, given oracle access to a function f, the algorithm outputs, with probability at least $2/3$, a value $\hat{\varepsilon}$ such that $\frac{1}{\eta} \cdot \varepsilon_{\mathcal{P}}(f) - \delta \leq \hat{\varepsilon} \leq \varepsilon_{\mathcal{P}}(f)$. If A works for all $\delta \in [0, 1)$, we call it an η- multiplicative distance approximation algorithm.*

Parnas et al. (2006) prove that the existence of a distance approximation algorithm for a property implies the existence of a tolerant tester for the same property. They also show that the existence of a fully tolerant tester for a property implies the existence of a distance approximation algorithm for the same property. Tolerant testers for many of the properties discussed in Section 3.2 are usually expressed as distance approximation algorithms. We now prove that the existence of a distance approximation algorithm for a property implies the existence of an erasure-resilient tester for that property (that works for a restricted range of parameters). Due to the equivalence between distance approximation and tolerant testing, the following theorem can be seen as a different version of Observation 3.9.2.

Observation 3.9.4. *Let $\eta \geq 1$ and $\delta \in [0, 1)$. Let A be an η-multiplicative δ-additive distance approximation algorithm for a property $\mathcal{P}$ of functions of the form $f : \mathcal{D} \to \mathcal{R}$. Then there exists an α-erasure-resilient ε-tester A' for $\mathcal{P}$ that makes the same number of queries as A and works for all $\varepsilon \in (0, 1), \alpha \in [0, 1)$ satisfying $\alpha + \varepsilon < 1$ and $\alpha < \frac{\varepsilon}{\eta} - \delta$.*

Proof. Fix an element $e \in \mathcal{R}$. As before, let f_e^r denote the completion of an α-erased function f where the erased points are assigned the value e.

Consider the following algorithm A'. The algorithm A', when given oracle access to an α-erased function $f : \mathcal{D} \to \mathcal{R} \cup \{\perp\}$, simulates oracle access to f_e^r and runs the tester A on f_e^r. Let $\hat{\varepsilon}$ denote the estimate that A computes at the end of its execution. If $\hat{\varepsilon} \leq \alpha$, the algorithm A' accepts. Otherwise, it rejects.

If an α-erased function f satisfies $\mathcal{P}$, then $\varepsilon_{\mathcal{P}}(f_e^r) \leq \alpha$. Since $\hat{\varepsilon} \leq \varepsilon_{\mathcal{P}}(f_e^r)$ with probability at least 2/3, the algorithm A' will accept f with probability at least 2/3.

If f is ε-far from $\mathcal{P}$, then every completion of f is ε-far from $\mathcal{P}$, and hence $\varepsilon_{\mathcal{P}}(f_e^r) \geq \varepsilon$. Since $\hat{\varepsilon} \geq \frac{\varepsilon_{\mathcal{P}}(f_e^r)}{\eta} - \delta \geq \frac{\varepsilon}{\eta} - \delta > \alpha$ with probability at least 2/3, the algorithm A'

will reject f with probability at least $2/3$. Note that the last inequality in the above expression follows from the restriction on α. $\square$

We now revisit the properties discussed in Section 3.2 for which distance approximation algorithms are known and apply Observation 3.9.4 to those algorithms and obtain erasure-resilient testers. The parameters of these testers are much worse than what we obtained in the technical sections of this chapter, especially in terms of the restrictions on α.

Corollary 3.9.5 (Saks & Seshadhri (2017))**.** *Let $c > 1$ be a constant. Let $\eta \in (1, 2)$. There exists an α-erasure-resilient ε-tester for monotonicity of real-valued functions over $[n]$ that works for all $\alpha \in [0, 1), \varepsilon \in (0, 1)$ such that $\alpha < \frac{\varepsilon}{\eta}$ and $\alpha + \varepsilon < 1$, with query complexity $O((\frac{1}{\varepsilon(\eta-1)})^{O(\frac{1}{\eta-1})} \cdot \log^c n)$.*

Corollary 3.9.6 (Fattal & Ron (2010))**.** *Let $\delta \in [0, 1]$. There exists an α-erasure-resilient ε-tester for monotonicity of functions from $[n]^d$ to a finite $R \subseteq \mathbb{R}$ that works for all $\alpha \in [0, 1), \varepsilon \in (0, 1)$ such that $\alpha + \varepsilon < 1$ and $\alpha < \frac{\varepsilon}{5d \log |R|} - \delta$, with query complexity $\tilde{O}\left(\frac{\log n}{\delta^3}\right)$.*

Corollary 3.9.7 (Fattal & Ron (2007))**.** *There exists an α-erasure-resilient ε-tester for convexity of real-valued functions over $[n]$ that works for all $\alpha \in [0, 1), \varepsilon \in (0, 1)$ such that $\alpha + \varepsilon < 1$ and $\alpha < \frac{\varepsilon}{25}$, with query complexity $\tilde{O}\left(\frac{\log n}{\varepsilon}\right)$.*

CHAPTER 4
Erasure-Resilient Property Testing Against Semi-Oblivious Adversary

In this chapter, we describe and study erasure-resilient property testing with a more powerful adversary.

4.1 MODEL DEFINITION AND OUR RESULTS

We first describe the erasure model that we consider and state our main results for erasure-resilient testing in that model.

Semi-Oblivious Erasure Oracles. An erasure oracle is one that can erase input values adversarially. It returns the symbol $\perp$ to an algorithm whenever the point queried by that algorithm is erased. The erasure oracle that we considered in Chapter 3 is one that erases points *obliviously*. In other words, all the erasures are made before the tester makes queries. In this chapter, we relax this obliviousness assumption and define the following erasure oracle.

Definition 4.1.1 (Semi-Oblivious Erasure Oracle)**.** *An erasure oracle is semi-oblivious if it makes at most* 1 *erasure adversarially before answering each query of the algorithm.*

A tester that accesses its inputs via a semi-oblivious erasure oracle is called a erasure-resilient tester against a semi-oblivious adversary.

Our Results. We investigate the complexity of erasure-resilient testing against a semi-oblivious adversary for sortedness of real-valued arrays as well as linearity of Boolean functions over the Boolean hypercube.

The problem of testing sortedness was already considered in Chapter 3. We prove that erasure-resilient testing sortedness of real-valued arrays against a semi-oblivious adversary is impossible.

Theorem 4.1.2. *For all $\varepsilon \in (0, \frac{1}{12}]$, no algorithm can ε-test sortedness of real-valued arrays accessed via a semi-oblivious erasure oracle.*

In contrast, as proven in Chapter 3, oblivious α-erasure-resilient ε-testing sortedness of real-valued arrays of length n has query complexity $\Theta(\log n)$ for all constant $\alpha \in [0,1), \varepsilon \in (0,1)$. This implies that sortedness is a natural property for which erasure-resilient testing against a semi-oblivious adversary is impossible, whereas oblivious erasure-resilient testing is easy.

Let $d \in \mathbb{N}$. Given $x, y \in \{0,1\}^d$, we use $x \oplus y$ to denote the vector $(x_1 + y_1, x_2 + y_2, \ldots, x_d + y_d)$, where $+$ denotes addition modulo 2. A function $f : \{0,1\}^d \to \{0,1\}$ is linear if for all $x, y \in \{0,1\}^d$, we have $f(x)+f(y) = f(x \oplus y)$. The following theorem shows that erasure-resiliently testing linearity against a semi-oblivious adversary can be done using a constant number of queries, independent of the size of the domain.

Theorem 4.1.3. *There exists an erasure-resilient ε-tester against a semi-oblivious adversary for linearity of functions $f : \{0,1\}^d \to \{0,1\}$ that makes $O(\frac{1}{\varepsilon^2})$ queries and works for all $\varepsilon \in \left[\Theta\left(\frac{1}{2^{d/4}}\right), 1\right)$.*

4.2 TESTING SORTEDNESS AGAINST SEMI-OBLIVIOUS ADVERSARY IS IMPOSSIBLE

In this section, we show that erasure-resiliently testing sortedness with access to a semi-oblivious erasure oracle is impossible and prove Theorem 4.1.2.

Proof of Theorem 4.1.2. By Yao's principle, it is enough to give a pair of distributions, one over sorted arrays and the other one over arrays that are far from sorted, and a strategy for the semi-oblivious erasure adversarial oracle such that there is no deterministic algorithm that can distinguish the distributions with high probability, when given access to them via that oracle.

Let $n \in \mathbb{N}$ be even. Consider the following distributions on arrays of length n.

Distribution $\mathcal{D}_{\mathsf{YES}}$: Independently for $i \in [n/2]$, the array values corresponding to indices $2i-1$ and $2i$ are:

- $2i-1$ and $2i-1$, respectively, with probability $1/3$.
- $2i-1$ and $2i$, respectively, with probability $1/3$.
- $2i$ and $2i$, respectively, with probability $1/3$.

Every array sampled from the distribution $\mathcal{D}_{\mathsf{YES}}$ is sorted.

Distribution $\mathcal{D}_{\mathsf{NO}}$: Independently for $i \in [n/2]$, the array values corresponding to indices $2i-1$ and $2i$ are:

- $2i$ and $2i-1$, respectively, with probability $1/3$.
- $2i-1$ and $2i$, respectively, with probability $2/3$.

For an array sampled from the distribution $\mathcal{D}_{\mathsf{NO}}$, the expected number of index pairs $(2i-1, 2i)$ such that the respective array values are $2i$ and $2i-1$ is equal to $n/6$. Let E denote the event that the number of such index pairs is at least than $n/12$. By a Chernoff bound, the probability of $\overline{E}$ is at most $\exp(-n/48)$. In other words, with probability at least $1 - \exp(-n/48)$, an array sampled from the distribution $\mathcal{D}_{\mathsf{NO}}$ is $1/12$-far from sorted.

Let $\mathcal{D}'_{\mathsf{NO}}$ denote the distribution $\mathcal{D}_{\mathsf{NO}}$ conditioned on E. Using Claim 4 by Raskhodnikova & Smith (2006), we get that the statistical distance between the distributions $\mathcal{D}'_{\mathsf{NO}}$ and $\mathcal{D}_{\mathsf{NO}}$ is at most $\frac{1}{1-\exp(-n/48)} - 1$.

Consider a deterministic adaptive two-sided error algorithm T for erasure-resilient $\frac{1}{12}$-testing sortedness against a semi-oblivious adversary. Assume that T is given access to an array sampled from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}'_{\mathsf{NO}}$ with equal probability, where the access is via a semi-oblivious erasure oracle $\mathcal{O}$. The oracle $\mathcal{O}$ behaves in the following way. For each $i \in [n/2]$, if T queries the point $2i - 1$ then the oracle erases the function value at the point $2i$; if T queries the point $2i$ then the oracle erases the function value at the point $2i - 1$. The oracle erases at most one array value before seeing a tester query.

The distribution of array values restricted to any particular index is identical in both the $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ distributions. Hence, the algorithm's view has the same distribution when the array is sampled from either $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$ distributions. Since the statistical distance between $\mathcal{D}'_{\mathsf{NO}}$ and $\mathcal{D}_{\mathsf{NO}}$ is at most $\frac{1}{1-\exp(-n/48)} - 1$, the statistical distance between the views of the algorithm on arrays sampled according to the $\mathcal{D}'_{\mathsf{NO}}$ and $\mathcal{D}_{\mathsf{YES}}$ distributions is at most $\frac{1}{1-\exp(-n/48)} - 1$, which is less than $1/3$ for all $n \geq 96$. Hence, the tester T cannot distinguish the distributions with probability at least $2/3$. The statement is true independent of the query complexity of T. □

4.3 ERASURE-RESILIENT LINEARITY TESTING AGAINST SEMI-OBLIVIOUS ADVERSARY

In this section, we describe our erasure-resilient linearity tester against a semi-oblivious adversary and prove Theorem 4.1.3.

Algorithm 4.1 describes our tester. The query complexity of the tester is evident from its description. The tester always accepts linear functions. To show that it rejects, with probability at least 2/3, every function that is ε-far from linear, we first prove Lemma 4.3.2. Lemma 4.3.2 is an extension of the following celebrated result by Bellare et al. (1996).

Algorithm 4.1 Erasure-Resilient Linearity Tester Against Semi-Oblivious Adversary

```
Input: $\varepsilon \in (0,1)$; oracle access to a function $f : \{0,1\}^d \to \{0,1\}$
1: repeat $\lceil \frac{2\ln 3}{\varepsilon^2} \rceil$ times:
2:     Query $f$ at $x, y, z \in_R \{0,1\}^d$.
3:     Uniformly at random execute either (a) or (b):
         (a) Query $f$ at $x \oplus z$ and reject if $f(x) + f(z) \neq f(x \oplus z)$.
         (b) Query $f$ at $y \oplus z$ and reject if $f(y) + f(z) \neq f(y \oplus z)$.
4: Accept.
```

Theorem 4.3.1. *If $f : \{0,1\}^d \to \{0,1\}$ is ε-far from linear, then*

$$\Pr_{x,y \in_R \{0,1\}^d} [f(x) + f(y) \neq f(x \oplus y)] \geq \varepsilon.$$

Lemma 4.3.2. *If $f : \{0,1\}^d \to \{0,1\}$ is ε-far from linear, then*

$$\Pr_{x,y,z \in_R \{0,1\}^d} [f(x) + f(z) \neq f(x \oplus z) \text{ and } f(y) + f(z) \neq f(y \oplus z)] \geq \varepsilon^2.$$

Proof. For $x, y \in \{0,1\}^d$, let $E_{x,y}$ denote the event that $f(x) + f(y) \neq f(x \oplus y)$. For $u \in \{0,1\}^d$, let $\varepsilon_u \in (0,1)$ denote the fraction of $v \in \{0,1\}^d$ such that $E_{u,v}$ holds.

For each fixed $z \in \{0,1\}^d$ and $x, y \in \{0,1\}^d$ sampled uniformly and independently at random, the events $E_{x,z}$ and $E_{y,z}$ are independent of each other.

We have,

$$\begin{aligned}
&\Pr_{x,y,z \in_R \{0,1\}^d} [E_{x,z} \cap E_{y,z}] \\
&= \mathbb{E}_{z \in \{0,1\}^d} \left[\Pr_{x,y \in_R \{0,1\}^d} [E_{x,z} \cap E_{y,z}] \right] \\
&= \mathbb{E}_{z \in \{0,1\}^d} \left[\Pr_{x \in_R \{0,1\}^d} [E_{x,z}] \cdot \Pr_{y \in_R \{0,1\}^d} [E_{y,z}] \right] \\
&= \mathbb{E}_{z \in \{0,1\}^d} [\varepsilon_z^2] \geq (\mathbb{E}_{z \in \{0,1\}^d} [\varepsilon_z])^2 \geq \varepsilon^2.
\end{aligned}$$

The first inequality follows from Jensen's inequality and the second inequality is a restatement of Theorem 4.3.1. □

The following lemma completes the proof of Theorem 4.1.3.

Lemma 4.3.3. *Let $d \geq 8$ be an integer. For all $\varepsilon \in \left[\Theta\left(\frac{1}{2^{d/4}}\right), 1\right)$, Algorithm 4.1 rejects functions $f : \{0,1\}^d \to \{0,1\}$ that are ε-far from linear with probability at least $2/3$.*

Proof. Let $f : \{0,1\}^d \to \{0,1\}$ be a function that is ε-far from linear. A triplet $(x, y, z) \in (\{0,1\}^d)^3$ is called a witness if x, y, z are all distinct and the events $f(x) + f(z) \neq f(x \oplus z)$ and $f(y) + f(z) \neq f(y \oplus z)$ hold. If there are no erasures, by Lemma 4.3.2, there are at least $\varepsilon^2 \cdot 2^{3d}$ triplets $(x, y, z) \in (\{0,1\}^d)^3$ such that the events $f(x) + f(z) \neq f(x \oplus z)$ and $f(y) + f(z) \neq f(y \oplus z)$ hold. The total number of triplets $(x, y, z) \in (\{0,1\}^d)^3$ that do not have distinct entries is $2^d + 3 \cdot 2^{2d}$. Hence, the number of witness triplets is at least $\varepsilon^2 \cdot 2^{3d} - 2^d - 3 \cdot 2^{2d}$.

Erasing a point $u \in \{0,1\}^d$ can damage at most $5 \cdot 2^{2d}$ witness triplets (x, y, z), since u can be either $x, y, z, x \oplus z$, or $y \oplus z$. The total number of queries that the tester makes is $4 \cdot \lceil \frac{2 \ln 3}{\varepsilon^2} \rceil \leq 4 \cdot \frac{4 \ln 3}{\varepsilon^2} \leq \frac{20}{\varepsilon^2}$, where the first inequality follows from the fact that for every $x > 1$, there is an integer between x and $2x$. Therefore, the number of remaining witness triplets is at least $\varepsilon^2 \cdot 2^{3d} - 2^d - 3 \cdot 2^{2d} - \frac{100}{\varepsilon^2} \cdot 2^{2d} \geq \frac{\varepsilon^2}{2} \cdot 2^{3d}$, where the inequality holds for all $\varepsilon \geq \frac{4}{2^{d/4}}$ and $d \geq 8$.

Therefore, in each iteration, the probability that the tester samples a (yet) non-erased witness triplet (x, y, z) in Step 1 is at least $\frac{\varepsilon^2}{2}$. Hence the probability that the tester detects a violation to linearity in one iteration is at least $\frac{\varepsilon^2}{2}$. Thus, the tester accepts with probability at most

$$\left(1 - \frac{\varepsilon^2}{2}\right)^{\lceil \frac{2 \ln 3}{\varepsilon^2} \rceil} \leq \frac{1}{3}$$

in $\lceil \frac{2\ln 3}{\varepsilon^2} \rceil$ iterations. $\square$

CHAPTER 5
Erasures versus Errors in Local Decoding

In this chapter, we compare the relative difficulty of local unique decoding and local list decoding in the presence of erasures and in the presence of errors.

The notion of locally decodable codes (LDCs) arose in the 1990s, motivated by numerous applications in complexity theory, such as program checking (Lipton (1990); Blum et al. (1993); Gemmell et al. (1991); Gemmell & Sudan (1992)), probabilistically checkable proofs (Babai et al. (1991); Arora & Safra (1998); Arora et al. (1998); Polishchuk & Spielman (1994)), derandomization (Babai et al. (1993); Sudan et al. (2001); Trevisan (2003)), and private information retrieval (Chor et al. (1998)). Locally decodable codes that work in the presence of errors have been extensively studied (Babai et al. (1991); Blum et al. (1993); Gemmell et al. (1991); Gemmell & Sudan (1992); Polishchuk & Spielman (1994); Beimel et al. (2002); Yekhanin (2008); Efremenko (2012); Dvir et al. (2011); Ben-Aroya et al. (2010a)). The related notion of locally list decodable codes (LLDCs) has also received a lot of attention (Goldreich & Levin (1989); Sudan et al. (2001); Gutfreund & Rothblum (2008); Ben-Aroya et al. (2010a); Kopparty & Saraf (2013); Kopparty (2015); Guo & Kopparty (2016); Gopi et al. (2018)) and found applications in cryptography (Goldreich & Levin (1989)), learning theory (Kushilevitz & Mansour (1993)), average-to-worst-case reductions (Lipton (1991); Cai et al. (1999); Goldreich et al. (2000a)), and hardness amplification and derandomization (Babai et al. (1993); Sudan et al. (2001)). The literature on decoding in the presence of erasures is too vast to survey here. *List* decoding in the presence of erasures (without the locality restriction) has been addressed by Guruswami (2003) and Guruswami & Indyk (2005). In particular, Guruswami (2003) constructed an asymptotically good family of binary linear codes that can be list decoded from an arbitrary fraction of erasures with lists of constant size. Even though decoding in the presence of erasures is an important and well established problem, to the best of our knowledge, local (unique and list) decoding from erasures has not been studied before[1].

5.1 MODEL DEFINITIONS AND OUR RESULTS

In this chapter, we restrict our attention to binary codes. A binary code is an infinite family of maps $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$. The parameter n is called the message length, N is the block length, and n/N is the rate of the code. Corruptions in codewords can either be in the form of erasures (missing entries, denoted by the symbol $\perp$) or in the form of errors (wrong values from $\mathbb{F}_2$).

Definition 5.1.1 (Locally Erasure-Decodable Codes (LEDCs))**.** *A code family $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is (α, q)-locally erasure-decodable if there exists an algorithm A that, given an index $i \in [n]$ and oracle access to an input word $w \in (\{\perp\} \cup \mathbb{F}_2)^N$ with at most an α fraction of erasures, makes at most q queries to w and outputs x_i with probability at least $\frac{2}{3}$.*

An (α, q)-locally decodable code (LDC) is defined similarly to an (α, q)-LEDC except that the input word w contains at most an α fraction of errors instead of erasures.

A local list decoder is a decoding algorithm that outputs a list of algorithms which give oracle access to decoded messages or, in other words *implicitly compute* the

[1] There is a related line of work on local list recovery (Hemenway et al. (2017); Gopi et al. (2018)), where codeword positions are associated with sets of symbols. The goal, given oracle access to such a codeword, is to output a list of codewords such that for each codeword in the list, the symbol at each position is equal to one of the symbols from the set associated with that position. In these terms, an erased codeword position corresponds to its associated set being equal to the alphabet.

decoded messages. This, and the notion of local list erasure-decoders are formalized in the following definitions.

Definition 5.1.2 (Implicit Computation)**.** *An algorithm A is said to implicitly compute $x \in \mathbb{F}_2^n$ if, for all $i \in [n]$, the algorithm A on input i, outputs the i^{th} bit of x.*

Definition 5.1.3 (Locally List Erasure-Decodable Codes (LLEDCs))**.** *A family of codes $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is (α, q, L)-locally list erasure-decodable if there exists a randomized algorithm A such that, for every $n \in \mathbb{N}$ and every $w \in (\mathbb{F}_2 \cup \{\perp\})^N$ with at most an α fraction of erasures, the algorithm A makes at most q queries to w and outputs a list of randomized algorithms $\{T_1, T_2, \ldots, T_L\}$ such that the following hold:*

1. *With probability at least $2/3$, for all $x \in \mathbb{F}_2^n$ such that $C_n(x)$ agrees with w on all nonerased bits, there exists an index $j \in [L]$ such that T_j with oracle access to w implicitly computes x.*

2. *For all $j \in [L]$ and $i \in [n]$, the expected number of queries that the algorithm T_j makes to w on input i is at most q.*

The definition of an (α, q, L)-LLDC is identical to Definition 5.1.3 except that the input word has no erasures, and the list is required to contain, with probability at least $2/3$, algorithms that implicitly compute messages corresponding to codewords disagreeing with the input word on at most an α fraction of bits.

An approximate local list erasure-decoder is identical to a local list erasure-decoder in all aspects except that the algorithms in its list are required to implicitly compute strings that are just "close" to the actual messages.

Definition 5.1.4 (Approximate Locally List Erasure-Decodable Codes (ALLEDCs))**.** *A family of codes $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is (α, q, L)-locally list erasure-decodable if there* *exists a randomized algorithm A such that, for every $n \in \mathbb{N}$ and every $w \in (\mathbb{F}_2 \cup \{\perp\})^N$ with at most an α fraction of erasures, the algorithm A makes at most q queries to w and outputs a list of randomized algorithms $\{T_1, T_2, \ldots, T_L\}$ such that the following hold:*

1. *With probability at least $2/3$, for all $x \in \mathbb{F}_2^n$ such that $C_n(x)$ agrees with w on all nonerased bits, there exists an index $j \in [L]$ such that T_j with oracle access to w implicitly computes a string $x' \in \mathbb{F}_2^n$ that is β-close to x.*

2. *For all $j \in [L]$ and $i \in [n]$, the expected number of queries that the algorithm T_j makes to w on input i is at most q.*

More formally, (α, β, q, L)-ALLEDCs are defined as (α, q, L)-LLEDCs in Definition 5.1.3, except that we replace "implicitly computes x" at the end of Item 1 with "implicitly computes ".

The definition of an (α, β, q, L)-approximate locally list decodable code (ALLDC) is identical to that of an (α, β, q, L)-ALLEDC except that the input word has no erasures, and the list is required to contain, with probability at least $2/3$, algorithms that implicitly compute strings that are β-close to messages corresponding to codewords which are α-close to the input word.

Our Results We now summarize our results and observations on local (unique and list) erasure-decoding.

Local Erasure-Decoding versus Local Decoding. We investigate the general relationship between the erasures and errors in the context of local unique and list decoding. We show that local (unique) decoding from erasures implies local (unique) decoding from errors, up to some loss in parameters.

We observe (based on an idea suggested to us by Venkatesan Guruswami) that an LDC is also locally erasure-decodable from (nearly) twice as many erasures.

Observation 5.1.5. *Every* (α, q)*-locally decodable code family* $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ *is also* $(2\alpha - \rho, O(q))$*-locally erasure-decodable, where* $\rho = O(\sqrt{\frac{\alpha}{N}})$.

We also observe the following about LLDCs and extend it to the case of ALLDCs.

Observation 5.1.6. *If a code family* $\{C_k : \mathbb{F}_2^k \to \mathbb{F}_2^n\}_{k\in\mathbb{N}}$ *is* (α, q, L)*-locally list decodable, it is also* $(2\alpha, 4q, 4L)$*-locally list erasure-decodable.*

Observation 5.1.7. *If a code family* $\{C_k : \mathbb{F}_2^k \to \mathbb{F}_2^n\}_{k\in\mathbb{N}}$ *is* (α, β, q, L)*-approximate locally list decodable, it is also* $(2\alpha, \beta, 4q, 4L)$*-approximate locally list erasure-decodable.*

We then show that constant-query LEDCs are constant-query locally decodable (up to constant loss in parameters).

Theorem 5.1.8. *For every* $\alpha \in [0, 1)$*, if a code family* $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in N}$ *is* (α, q)*-locally erasure-decodable, then it is* $(\frac{\alpha}{O(3^q)}, O(3^q))$*-locally decodable.*

To prove Theorem 5.1.8, we start with a local erasure-decoder for $\{C_n\}_{n\in\mathbb{N}}$ and transform it to be a nonadaptive and smooth local erasure-decoder. An algorithm is nonadaptive if its queries do not depend on the answers to the previous queries. A decoding algorithm is smooth if it decodes uncorrupted codewords by querying nearly uniformly distributed codeword indices. We first make the local erasure-decoder for $\{C_n\}_{n\in\mathbb{N}}$ nonadaptive. We then show that every nonadaptive decoding algorithm for an LEDC can be transformed into a smooth decoding algorithm. We then use this 'smoothness' feature to show that the code family is locally decodable from a smaller fraction of errors than erasures.

The technique outlined above cannot be directly used to obtain an analog of Theorem 5.1.8 for the case of local list decoding since the notion of smoothness (the way we define it for use in our transformation) does not make sense in the local list decoding setting. Smooth local decoding assumes oracle access to an uncorrupted codeword and the goal is to decode the message by making nearly uniformly distributed queries. Local list decoding, however, is relevant in the setting that a codeword has a higher number of corrupt bits than the unique decoding radius.

Local List Erasure-Decoding of Hadamard Code. Our next result is about the local list erasure-decodability of the Hadamard code. The celebrated Goldreich-Levin theorem (Goldreich & Levin (1989)) states that the Hadamard code, defined next, is an LLDC that has an efficient decoder. We prove an analogue of the Goldeich-Levin Theorem (Goldreich & Levin (1989)) for corruptions in the form of erasures.

Definition 5.1.9 (Hadamard code)**.** *For $a \in \mathbb{F}_2^n$, let $H_a : \mathbb{F}_2^n \to \mathbb{F}_2$ be defined as follows: $H_a(x) = \bigoplus_{i\in[n]} a_i \cdot x_i$ for all $x \in \mathbb{F}_2^n$. The Hadamard code, denoted by $\{\mathcal{H}_n : \mathbb{F}_2^n \to \mathbb{F}_2^{2^n}\}_{n\in\mathbb{N}}$, is such that for $a \in \mathbb{F}_2^n$, the encoding $\mathcal{H}_n(a)$ is the string of evaluations of H_a over $\mathbb{F}_2^n$.*

Theorem 5.1.10 (Local List Erasure-Decoder for Hadamard)**.** *There is an $\big(\alpha, O(\frac{\alpha}{1-\alpha}), O(\frac{\alpha}{1-\alpha})\big)$-local list erasure-decoder for the Hadamard code that works for every $\alpha \in [0, 1)$.*

The Goldreich-Levin theorem holds for any fraction of errors in $[0, 1/2)$. In contrast, our local list erasure-decoder works for any fraction of erasures less than 1. However, it is impossible to decode the Hadamard code in the presence of 1/2 fraction of errors because every Hadamard codeword has relative distance at most 1/2 from the all-zero codeword. Another improvement in Theorem 5.1.10 as compared to Golreich-Levin is in the list size and the query complexity: from $\Theta(\frac{1}{(1/2-\alpha)^2})$ to $O(\frac{\alpha}{1-\alpha})$. Such an improvement is impossible if we are decoding against errors as opposed to erasures. Specifically, for the list size, Blinovsky (1986) and Guruswami & Vadhan (2010) show that every list decoder for every binary code that is list decodable in the presence of an α fraction of errors must output lists of size $\Omega(\frac{1}{(1/2-\alpha)^2})$. No such general lower bounds are known for query complexity. Recently, Grinberg et al. (2018) (implicitly) prove lower bounds of $\Omega(\frac{1}{(1/2-\alpha)^2})$ and $\Omega(\frac{1}{1-\alpha})$ on the query complexity of local list decoding and local list erasure-decoding, respectively, of binary codes with subexponential rate. In particular, their lower bounds do not apply to Hadamard.

By combining Observation 5.1.6 with the Goldreich-Levin theorem, one can obtain a local list erasure-decoder for the Hadamard code that works for every $\alpha \in [0, 1)$ and has list size and query complexity $\Theta(\frac{1}{(1-\alpha)^2})$. However, we obtain strictly better list size and query complexity in Theorem 5.1.10.

5.2 LOCAL ERASURE-DECODING

In this section, we prove Theorem 5.1.8 and our observations that if a code is locally decodable, it is also locally erasure-decodable up to (nearly) twice as many erasures.

Definition 5.2.1 (Smooth Locally Decodable Codes). *A code family $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is (q, η)-smooth locally decodable if there exists a $(0, q)$-local erasure-decoder A (see Definition 5.1.1) that, given oracle access to an uncorrupted codeword $w \in \mathbb{F}_2^N$, and an input $i \in [n]$, is such that, for all $j \in [N]$, the probability that A queries j is at most η.*

It is easy to see that the following two claims imply Theorem 5.1.8.

Claim 5.2.2. *For every $\alpha \in [0,1)$, if a code family $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is (α, q)-locally erasure-decodable, then $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is (q', η)-smooth locally decodable, where $q' = 3^q$, and $\eta = \frac{q'}{\alpha N}$.*

Claim 5.2.3. *For every $\alpha \in [0,1)$, if a code family $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is $(q, \frac{q}{\alpha\cdot N})$-smooth locally decodable, then $\{C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N\}_{n\in\mathbb{N}}$ is $(\frac{\alpha}{O(q)}), O(q)$-locally decodable.*

Proof of Claim 5.2.2. Let A be an (α, q)-local erasure-decoder for C_n. Since A could be adaptive, for every choice of random coins, the execution of A can be represented as a ternary tree, where each node represents a query. The root represents the first query made by A. The three children of a non-leaf node u represent the next points that A will query for the cases that the answers to the query u are $0, 1$, or $\perp$. The size of this tree is at most 3^q. Consider an algorithm A_1 that, after having generated its random string $r \in \{0,1\}^*$, queries all the points in the tree of execution of A on the string r. After obtaining the answers to its queries, A_1 outputs the value at the end of the root-to-leaf path that matches with the actual query answers. Note that there is exactly one such path. Therefore, A_1 is a nonadaptive local erasure-decoder for C_n that makes $q' = 3^q$ queries and has the same success probability as A.

We now use A_1 to construct A_2, a $(q', \frac{q'}{\alpha N})$-smooth local decoder for C_n. Consider an uncorrupted codeword $w = C_n(x)$ for $x \in \mathbb{F}_2^n$. For each $i \in [n]$, let S_i denote the set consisting of indices in $[N]$ that get queried by A_1 (on input i) with probability more than $\frac{q'}{\alpha N}$. Since $\sum_{j\in[N]} \Pr[A_1^{C_n(x)}(i) \text{ queries } j] = q'$, we have $|S_i| \leq \alpha \cdot N$. On input $i \in [n]$ and oracle access to $w = C_n(x)$, the algorithm A_2 simulates A_1 in the following way. If A_1 queries $j' \in S_i$, the algorithm A_2 does not query j' and assumes that $w[j'] = \perp$. Thus, A_2 is a $(q', \frac{q'}{\alpha N})$-smooth local decoder for C_n. □

Proof of Claim 5.2.3. Consider a $(q, \frac{q}{\alpha N})$-smooth local decoder A for C_n. We will construct an $(\frac{\alpha}{12q}, 72q)$-local decoder A' for C_n. Algorithm A', on input $i \in [n]$ and oracle access to a word w with at most $\frac{\alpha}{12q}$ fraction of errors, performs 72 independent repetitions of A and outputs the majority value output among all the iterations.

Let $x \in \mathbb{F}_2^n$ be such that $y = C_n(x)$ is the codeword closest to w. If A is run on input i with oracle access to y, then for at least $\frac{2}{3}$ fraction of the sequences of its random coin tosses, A returns x_i correctly. When A is run on input i with oracle access to w, by the union bound and the smoothness of A, at most $\frac{\alpha}{12q} \cdot N \cdot \frac{q}{\alpha N} = \frac{1}{12}$ fraction of sequences of its random coin tosses result in an erroneous position being queried. Hence, the probability that A, on input i and oracle access to w, returns x_i correctly is at least $\frac{2}{3} - \frac{1}{12}$. Hence, the probability that A' outputs x_i correctly is at least $2/3$. The query complexity of A' is $72q$. □

Even though the above technique cannot be directly used to obtain an analog of Theorem 5.1.8 for the case of local list decoding (see the discussion after the statement of Theorem 5.1.8), by making a soundness assumption on the list returned by a local list decoder, we can get an analog of Theorem 5.1.8 for the case of local list decoding. In particular, assume that an LLEDC $\mathcal{C}$ has a decoder such that every algorithm in the output list corresponds to a valid message (whose encoding is close to the codeword). If the fraction of erasures is so *small* that there is only one codeword that agrees with the input word on all nonerased bits, we can use the sound decoder for local unique decoding. This would imply the existence of a local unique decoder for $\mathcal{C}$ that decodes from an *even smaller* fraction of errors (by Theorem 5.1.8). One can then apply a transformation by Ben-Aroya et al. (2010b) and show that there is another code $\mathcal{C}'$ (with a slightly worse rate) that is an LLDC.

We now prove Observation 5.1.5.

Proof of 5.1.5. Consider an (α, q)-local decoder A for $C_n : \mathbb{F}_2^n \to \mathbb{F}_2^N$. Let $w \in (\mathbb{F}_2 \cup \{\perp\})^N$ be a codeword with at most $(2\alpha - \rho)N$ erasures. Consider algorithm

A' that, on input $i \in [n]$ and oracle access to w, runs A on input $i \in [n]$ and oracle access to w', where w' is generated on the fly by filling in the erased bits of w with 0 or 1 u.a.r. The expected Hamming distance of w' to the code is at most $\alpha N - \frac{\varrho}{2} N$. By a Chernoff bound, the probability that the Hamming distance of w' to the code is more than αN is at most $\frac{1}{12}$. The probability of failure of A' is at most $\frac{5}{12}$. One can amplify the success probability to $2/3$ by performing 72 independent repetitions of A' and outputting the majority answer. □

We also provide the proof of that every LLDC that works in the presence of errors also works in the presence of twice as many erasures (with the same parameters up to constant factors). For the sake of completeness, we also provide a proof for a similar statement for approximate locally list erasure-decodable codes, even though the two proofs are nearly identical.

Proof of Observation 5.1.6. Consider a codeword $w \in (\mathbb{F}_2 \cup \{\perp\})^n$ with at most 2α fraction of erasures. Let A be an (α, q, L)-local list decoder for C_k. Assume without loss of generality that the success probability of A is at least $4/5$. This can be ensured by running A twice and outputting the concatenation of lists obtained in both iterations. The local list erasure-decoder A' for C_k first runs A on the word w_0 obtained by replacing each erasure in w with a 0, and then on the word w_1 obtained by replacing each erasure in w with a 1. The list output by algorithm A' is the concatenation of lists output by A in these two executions. Let E_1 be the event that the first execution of A succeeds and E_2 be the event that the second execution of A succeeds. Each codeword $w' = C_k(y')$ that agrees with w on all the nonerased points agrees with either w_0 or w_1 in at least $1 - \alpha$ fraction of points. If $E_1 \cap E_2$ holds, there exists an algorithm in the list output by A' that implicitly computes (see Definition 5.1.2) y'. The probability of failure of A' is at most $\Pr[\overline{E_1} \cup \overline{E_2}] \leq \frac{2}{5}$.

Hence, A' is a $(2\alpha, 4q, 4L)$-approximate local list erasure-decoder for C_k. $\square$

Proof of Observation 5.1.7. Consider a codeword $w \in (\mathbb{F}_2 \cup \{\perp\})^n$ with at most 2α fraction of erasures. Let A be an (α, β, q, L)-approximate local list decoder for C_k. Assume without loss of generality that the success probability of A is at least 4/5. This can be ensured by running A twice and outputting the concatenation of lists obtained in both iterations. The approximate local list erasure-decoder A' for C_k first runs A on the word w_0 obtained by replacing each erasure in w with a 0, and then on the word w_1 obtained by replacing each erasure in w with a 1. The list output by algorithm A' is the concatenation of lists output by A in these two executions. Let E_1 be the event that the first execution of A succeeds and E_2 be the event that the second execution of A succeeds. Each codeword $w' = C_k(y')$ that agrees with w on all the nonerased points agrees with either w_0 or w_1 in at least $1 - \alpha$ fraction of points. If $E_1 \cap E_2$ holds, there exists an algorithm in the list output by A' that implicitly computes (see Definition 5.1.2) a string y'' that is β-close to y'. The probability of failure of A' is at most $\Pr[\overline{E_1} \cup \overline{E_2}] \leq \frac{2}{5}$. Hence, A' is a $(2\alpha, \beta, 4q, 4L)$-approximate local list erasure-decoder for C_k. $\square$

5.3 LOCAL LIST ERASURE DECODING OF HADAMARD CODE

As mentioned earlier, Theorem 5.1.10 is an analogue of the Goldreich-Levin Theorem for the case of erasures. We follow the style of the proof of the Goldreich-Levin theorem given in a tutorial by Trevisan (2004) on the applications of coding theory to complexity.

Proof of Theorem 5.1.10. For $b_1, b_2 \in \mathbb{F}_2$, let $b_1 \oplus b_2$ denote the XOR of b_1 and b_2. For vectors $x, y \in \mathbb{F}_2^n$, let $x \odot y$ denote the bitwise XOR of x and y. Let $e_k \in \mathbb{F}_2^n$ denote the kth standard basis vector. A codeword of the Hadamard code $\mathcal{H}_n$ (see Definition 5.1.9)

Algorithm 5.1 Local List Erasure-Decoder for the Hadamard code

```
Input: $\alpha \in [0,1)$; oracle access to $\alpha$-erased linear function $f : \mathbb{F}_2^n \to \mathbb{F}_2 \cup \{\perp\}$
    ▷ Let $t \leftarrow \lceil \log_2(\frac{12\alpha}{1-\alpha}) \rceil$.
1:  Sample and query $z_1, z_2, \ldots, z_t \in \mathbb{F}_2^n$ uniformly and independently at random.
    ▷ Let $z_S \leftarrow \bigodot_{i \in S} z_i$ for all nonempty $S \subseteq [t]$. Let $z_\phi \leftarrow \vec{0}$. Let $B \leftarrow \{i \in [t] : f(z_i) = \perp\}$.
2:  for all $b_1, b_2, \ldots, b_{|B|} \in \{0,1\}$ do define
    ▷ Description of the local decoder $T_{b_1,\ldots,b_{|B|}}$ follows.
3:      function $A_{b_1,\ldots,b_{|B|}}$
4:          input: $x \in \mathbb{F}_2^n$; oracle access to $f : \mathbb{F}_2^n \to \mathbb{F}_2 \cup \{\perp\}$
5:          for all $S \subseteq [t]$ do
6:              if $f(x \odot z_S) \neq \perp$ then
                    return $(\bigoplus_{j \in S \cap B} b_j) \oplus (\bigoplus_{j \in S \cap ([t] \setminus B)} f(z_j)) \oplus f(x \odot z_S)$.
7:          return $\perp$.
8:      function $T_{b_1,\ldots,b_{|B|}}$
9:          input: $k \in [n]$; oracle access to $f : \mathbb{F}_2^n \to \mathbb{F}_2 \cup \{\perp\}$
10:         repeat
11:             Pick $y \in \mathbb{F}_2^n$ uniformly and independently at random.
12:             $u \leftarrow A_{b_1,\ldots,b_{|B|}}(y \odot e_k)$, $v \leftarrow A_{b_1,\ldots,b_{|B|}}(y)$.
13:             if $v \neq \perp$ and $u \neq \perp$ then return $u \oplus v$.
14: return the descriptions of $T_{b_1,\ldots,b_{|B|}}$ for all $b_1, b_2, \ldots, b_{|B|} \in \{0,1\}$.
```

is the string of all evaluations of a linear function mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2$. A function $f : \mathbb{F}_2^n \to \mathbb{F}_2 \cup \{\perp\}$ is α-erased, if f evaluates to $\perp$ on at most an α fraction of its domain. Our local list erasure-decoder, described in Algorithm 5.1, gets a parameter $\alpha \in [0, 1)$ as its input and has oracle access to an α-erased linear function $f : \mathbb{F}_2^n \to \mathbb{F}_2 \cup \{\perp\}$ (or, equivalently, oracle access to an α-erased codeword of $\mathcal{H}_n$).

We now analyze Algorithm 5.1. For a string $a \in \mathbb{F}_2^n$, the function $H_a : \mathbb{F}_2^n \to \mathbb{F}_2$ denotes the Hadamard encoding of a (see Definition 5.1.9). We will show that, with probability at least $2/3$, for every $a \in \mathbb{F}_2^n$ such that the functions H_a and f agree with each other on all the nonerased points, one of the local decoders output by Algorithm 5.1 implicitly computes a (see Definition 5.1.2).

There exists some iteration of Step 2 of Algorithm 5.1 such that $b_i = H_a(z_i)$ for all $i \in B$. Let T and A denote the algorithms whose descriptions are generated in Steps 8 and 3 of this iteration, respectively.

First, we show that for x distributed uniformly in $\mathbb{F}_2^n$, the algorithm A on input x, returns $H_a(x)$ with probability at least $2/3$. Consider the first set $S' \subseteq [t]$ (in the order that A considers sets) such that $f(x \odot z_{S'}) \neq \perp$. According to the description of A,

$$\begin{aligned} A(x) &= \left(\bigoplus_{j \in S' \cap B} b_j \right) \oplus \left(\bigoplus_{j \in S' \cap ([t] \setminus B)} f(z_j) \right) \oplus f(x \odot z_{S'}) \\ &= \left(\bigoplus_{j \in S' \cap B} H_a(z_j) \right) \oplus \left(\bigoplus_{j \in S' \cap ([t] \setminus B)} H_a(z_j) \right) \oplus H_a(x \odot z_{S'}) \\ &= \left(\bigoplus_{j \in S'} H_a(z_j) \right) \oplus H_a(x) \oplus \left(\bigoplus_{j \in S'} H_a(z_j) \right) = H_a(x). \end{aligned}$$

It remains to show that, with probability at least $2/3$, there exists some set $S \subseteq [t]$ such that $f(x \odot z_S) \neq \perp$. Let $\alpha^\star \leq \alpha$ denote the fraction of erasures in f. For each $S \subseteq [t]$, we have that $f(x \odot z_S) \neq \perp$ with probability $1 - \alpha^\star$, since x (and therefore, $x \odot z_S$) is uniformly distributed in $\mathbb{F}_2^n$. Define indicator random variables $Z_S = \mathbb{1}(f(x \odot z_S) \neq \perp)$ for $S \subseteq [t]$ and let $Z = \sum_{S \subseteq [t]} Z_S$. The random variable Z is equal to the number of nonerased values among $f(x \odot z_S)$ for $S \subseteq [t]$. The event that $\forall S \subseteq [t], f(x \odot z_S) = \perp$ is equivalent to the event that $Z < 1$.

For each $S \subseteq [t]$, we have $\mathbb{E}[Z_S] = 1 - \alpha^\star$. Therefore, by the linearity of expectation,

$$\mathbb{E}[Z] = \sum_{S \subseteq [t]} \mathbb{E}[Z_S] = 2^t(1 - \alpha^\star).$$

For every two nonempty sets $R, S \subseteq [t]$ such that $R \neq S$, the vectors z_R and z_S are independently and uniformly distributed in $\mathbb{F}_2^n$. Thus, the collection $\{x \odot z_S | S \subseteq [t]\}$ is pairwise independent, and hence the random variables Z_S for $S \subseteq [t]$ are also pairwise independent. Now, for each $S \subseteq [t]$, we have $\mathrm{Var}(Z_S) = (1 - \alpha^\star) \cdot \alpha^\star$, and by the pairwise independence,

$$\mathrm{Var}[Z] = \sum_{S \subseteq [t]} \mathrm{Var}[Z_S] = 2^t \cdot \alpha^\star(1 - \alpha^\star).$$

Applying the Chebyshev's inequality,

$$\begin{aligned}
\Pr[Z < 1] &= \Pr[\mathbb{E}[Z] - Z > \mathbb{E}[Z] - 1] \\
&\leq \Pr[\mathbb{E}[Z] - Z \geq 2^t \cdot (1 - \alpha^\star) - 1] \\
&\leq \Pr\left[\mathbb{E}[Z] - Z > \frac{2^t \cdot (1 - \alpha^\star)}{2}\right] \leq \frac{4\mathrm{Var}(Z)}{(1 - \alpha^\star)^2 \cdot (2^t)^2} \\
&\leq \frac{4\alpha^\star}{(1 - \alpha^\star) \cdot 2^t} \leq \frac{4\alpha}{(1 - \alpha) \cdot 2^t} \leq \frac{1}{3}.
\end{aligned}$$

The last inequality follows from our setting of t. Therefore, for x distributed uniformly in $\mathbb{F}_2^n$, algorithm A on input x, returns $H_a(x)$ with probability at least $\frac{2}{3}$.

Finally, we prove that T implicitly computes $a \in \mathbb{F}_2^n$ and that the expected number of queries that T makes to f is $O(\frac{\alpha}{1-\alpha})$. It is clear that the output of T on input $k \in [n]$ is always $a[k] = H_a(y \odot e_k) \oplus H_a(y) = H_a(e_k)$. The number of queries made by T to A is a geometric random variable with success probability at least $1/3$. Hence, the expected number of queries made by T to A is at most 3. Since the query complexity of A is at most 2^t, the expected number of queries made to f in one invocation of T is $O(2^t)$, which is equal to, $O(\frac{\alpha}{1-\alpha})$. The number of algorithms whose descriptions are generated is also at most 2^t, which is also $O(\frac{\alpha}{1-\alpha})$. $\square$

CHAPTER 6
Separating Standard, Erasure-Resilient, and (Error) Tolerant Property Testing

In this chapter, we study the relationship of erasure-resilient property testing (Dixit et al. (2018)) with standard property testing (Goldreich et al. (1998); Rubinfeld & Sudan (1996)) and tolerant property testing (Parnas et al. (2006)).

6.1 OUR RESULTS

Separating Standard and Erasure-Resilient Testing. We provide (in Section 6.2) the following separation between our erasure-resilient model and the standard model. Specifically, we prove the existence of a property that can be tested with a constant number of queries in the standard model, but has query complexity nearly linear in the length of the input in the erasure-resilient model.

Theorem 6.1.1. *There exists a property $\mathcal{P}$ and constants $\varepsilon, \alpha \in (0,1)$ satisfying $\alpha + \varepsilon < 1$, such that,*

- *$\mathcal{P}$ can be ε-tested using a constant number of queries,*
- *every α-erasure-resilient ε-tester for $\mathcal{P}$ on inputs of length n has query complexity $\tilde{\Omega}(n)$.*

Separating Erasure-Resilient and Tolerant Testing. Recall that a tolerant tester for a property $\mathcal{P}$, given two parameters $\varepsilon_1, \varepsilon_2 \in (0,1)$, where $\varepsilon_1 < \varepsilon_2$, is required to, with probability at least $2/3$, accept inputs that are ε_1-close to $\mathcal{P}$ and reject inputs that are ε_2-far from $\mathcal{P}$.

Comparison of Parameters. While comparing the two models, it is appropriate to compare $(\alpha, \alpha+\varepsilon)$-tolerant testing of a property $\mathcal{P}$ with α-erasure-resilient ε-testing of $\mathcal{P}$ for the same values of $\alpha \in [0,1)$ and $\varepsilon \in (0,1)$ such that $\alpha+\varepsilon < 1$. The parameter α in both models is an upper bound on the fraction of corruptions (erasures, or errors) that an adversary can make to an input. An α-erasure-resilient ε-tester rejects with probability at least $\frac{2}{3}$ if, for every way of completing an input string, one needs to change at least an ε fraction of the input to make it satisfy $\mathcal{P}$. Similarly, an $(\alpha, \alpha+\varepsilon)$-tolerant tester rejects with probability at least $\frac{2}{3}$ if, for every way of *correcting* an α fraction of the input values, one needs to change at least an additional ε fraction of the input to make it satisfy $\mathcal{P}$.

Intuitively, the relationship of our erasure-resilient model to tolerant testing is akin to the relationship between error-correcting codes that withstand erasures and error-correcting codes that withstand general errors.

Hadamard-based Separation. The following theorem states that there exists a property that is erasure-resiliently testable but is not tolerantly testable. This proves that tolerant testing is, in general, harder problem than erasure-resilient testing.

Theorem 6.1.2 (Hadamard-based Separation)**.** *There exists a property $\mathcal{P}'$ and constants $\alpha, \varepsilon \in (0,1)$ such that*

- *$\mathcal{P}'$ can α-erasure-resiliently ε-tested using a constant number of queries;*
- *every $(\alpha, \alpha+\varepsilon)$-tolerant tester for $\mathcal{P}'$, on inputs of length n, has query complexity $\tilde{\Omega}(\log n)$.*

Strengthened Separation. We obtain a separation better than Theorem 6.1.2 with the help of a variant of LLEDCs, called approximate locally list erasure-decodable codes (ALLEDCs, see Definition 5.1.4).

We use the Observation 5.1.7 that every (α, β, q, L)-ALLDC is also a $(2\alpha, \beta, 4q, 4L)$-ALLEDC, and combine it with existing constructions for ALLDCs due to Impagliazzo et al. (2010); Ben-Aroya et al. (2010b) to obtain efficient ALLEDCs. We use them and get our strengthened separation.

Theorem 6.1.3 (Strengthened Separation)**.** *There exists a property $\mathcal{P}''$ and constants $\alpha, \varepsilon \in (0,1)$ such that*

- *$\mathcal{P}''$ is α-erasure-resiliently ε-testable;*
- *every $(\alpha, \alpha+\varepsilon)$-tolerant tester for $\mathcal{P}''$, on inputs of length n, has query complexity $n^{\Omega(1)}$.*

6.2 SEPARATING ERASURE-RESILIENT AND STANDARD TESTING

In this section, we describe a property that can be tested in the standard model using a constant number of queries, but for which every erasure-resilient tester has query complexity nearly linear in the input length and prove Theorem 6.1.1. The following theorem implies Theorem 6.1.1.

Theorem 6.2.1. *There exists a property $\mathcal{P}$ and a constant $\varepsilon^\star \in (0,1)$ such that,*

- *For every constant $\varepsilon \in (0,1)$, the property $\mathcal{P}$ can be ε-tested using $O(1/\varepsilon)$ queries,*
- *For every constants $\varepsilon \in (0, \frac{\varepsilon^\star}{2}]$ and $\alpha \in (0,1)$ satisfying $\alpha + \varepsilon < 1$, every α-erasure-resilient ε-tester for $\mathcal{P}$ on inputs of length N has query complexity $\tilde{\Omega}(N)$.*

6.2.1 Separating Property $\mathcal{P}$: Hard CNF property and PCPs of proximity

The property $\mathcal{P}$ is defined in terms of a property $\mathcal{R}$ that is hard to test in the standard property testing model (Goldreich et al. (1998); Rubinfeld & Sudan (1996)) and a probabilistically checkable proof system (PCP of proximity(Ergün et al. (2004); Ben-Sasson et al. (2006); Dinur & Reingold (2006))) for the problem of testing $\mathcal{R}$. We discuss them below. As mentioned earlier, the idea of using PCPs of proximity in separating two property testing models comes from the work of Fischer & Fortnow (2006).

We now describe the property $\mathcal{R}$ that is hard to test in the standard model. Given $n \in \mathbb{N}$ and a Boolean formula ϕ over n variables, let $\mathcal{R}_\phi \subseteq \{0,1\}^n$ denote the set of all satisfying assignments to ϕ, represented as n-bit strings. Ben-Sasson, Harsha, & Raskhodnikova (2005) showed that for infinitely many $n \in \mathbb{N}$, there exists a 3CNF formula ϕ_n on n variables such that every tester for $\mathcal{R}_{\phi_n}$ requires $\Omega(n)$ queries.

Lemma 6.2.2 (Ben-Sasson, Harsha, & Raskhodnikova (2005))**.** *There exists a parameter $\varepsilon^\star \in (0,1)$ and a countably infinite set $\aleph \subseteq \mathbb{N}$ such that for all $n \in \aleph$, there exists a 3CNF formula ϕ_n with n variables and $\Theta(n)$ clauses such that every $\varepsilon^\star$-tester for $\mathcal{R}_{\phi_n}$ has query complexity $\Omega(n)$.*

An important ingredient in the description of the separating property $\mathcal{P}$ is a probabilistically checkable proof system for property testing problems. The notion of proof assisted property testing was introduced by Ergün, Kumar, & Rubinfeld (2004). Ben-Sasson et al. (2006) and Dinur & Reingold (2006) defined and studied a special case of proof-assisted property testers called PCPs of proximity (or alternatively, assignment testers). PCPs of proximity were further studied by Dinur (2007) and Meir (2014, 2016).

Definition 6.2.3 (PCP of proximity (Ergün et al. (2004); Ben-Sasson et al. (2006); Dinur & Reingold (2006))). *Given a property $P_n \subseteq \{0,1\}^n$, the PCP of proximity (PCPP) for P_n is a randomized algorithm V that takes a parameter $\varepsilon \in (0,1]$, gets oracle access to a string $y \circ \pi$, where $y \in \{0,1\}^n$ is the input and $\pi \in \{0,1\}^m$ is the proof, and satisfies the following:*

- *if $y \in P_n$, then, for some π, the algorithm V always accepts $y \circ \pi$;*
- *if y is ε-far from P_n, then, for every π, the algorithm V rejects $y \circ \pi$ with probability at least $\frac{2}{3}$.*

A result of (Dinur, 2007, Corollary 8.4) implies that there are *efficient* PCPPs (over a small constant alphabet Σ) for testing properties (over Σ) that are decidable using polynomial-sized circuits. The following restatement of this result is obtained by representing the symbols in Σ using the binary alphabet.

Lemma 6.2.4 (Dinur (2007)). *If $P_n \subseteq \{0,1\}^n$ is a property decidable by a circuit of size $s(n)$, then there exists a randomized algorithm V' that gets oracle access to a string $y \circ \pi \in \{0,1\}^*$, where $y \in \{0,1\}^n$ is the input and π is a proof of length at most $s(n) \cdot \text{polylog}\, s(n)$, and satisfies the following:*

- *if $y \in P_n$, then for some proof π, the algorithm V' always accepts $y \circ \pi$;*
- *if $y \notin P_n$, then for every π, the algorithm V' rejects $y \circ \pi$ with probability proportional to the relative Hamming distance of y from P_n.*

Moreover, the queries of V' are nonadaptive.

An algorithm guaranteed by Lemma 6.2.4 for a property P can be converted to an efficient PCPP for P by simply repeating the former algorithm sufficiently many times.

Lemma 6.2.5 (Dinur (2007))**.** *If $P_n \subseteq \{0,1\}^n$ is a property decidable by a circuit of size $s(n)$, then there exists a PCPP V that works for every $\varepsilon \in (0,1]$, uses a proof of length at most $s(n) \cdot \operatorname{polylog} s(n)$, and has query complexity $O(\frac{1}{\varepsilon})$. Moreover, the queries of V are nonadaptive.*

Claim 6.2.6 uses Lemma 6.2.5 in conjunction with the fact that the property $\mathcal{R} = \{\mathcal{R}_{\phi_n}\}_{n\in\aleph}$ can be decided using linear-sized circuits.

Claim 6.2.6. *There exists a constant $c > 0$ such that for every large enough $n \in \mathbb{N}$, there exists a PCPP V for the property $\mathcal{R}_{\phi_n}$ that works for all $\varepsilon \in (0,1]$, uses a proof of length at most $cn \cdot \operatorname{polylog} n$, and has query complexity $O(\frac{1}{\varepsilon})$.*

Proof. One can observe that for all $n \in \aleph$, the circuit complexity of deciding $\mathcal{R}_{\phi_n}$ (described in Lemma 6.2.2) is $O(n)$. In other words, there exists a c'' such that for every large enough n, the property $\mathcal{R}_{\phi_n}$ can be decided using a circuit of size at most $c''n$. The claim follows by plugging this fact into Lemma 6.2.5. □

The following is the definition of our separating property $\mathcal{P}$. At a high level, the definition says that, for all $n \in \aleph$, a string of length $O(n \cdot \operatorname{polylog} n)$ satisfies $\mathcal{P}$ if its first part is the repetition of a string y satisfying $\mathcal{R}_{\phi_n}$, and the second part is a proof π that makes the algorithm V in Claim 6.2.6 accept.

Definition 6.2.7 (Separating Property $\mathcal{P}$)**.** *Let $\varepsilon^\star \in (0,1)$ and $\aleph \subseteq \mathbb{N}$ be from Lemma 6.2.2. For $n \in \aleph$, let $p(n) \le cn \cdot \operatorname{polylog} n$ denote the length of proof that the algorithm V in Claim 6.2.6 has oracle access to. A string $x \in \{0,1\}^N$ of length $N = (\log n + 1) \cdot p(n)$ satisfies $\mathcal{P}$ if the following conditions hold:*

1. *The first $p(n) \cdot \log n$ bits of x (called the input part of x) consist of $\frac{p(n)\cdot\log n}{n}$ repetitions of a string $y \in \mathcal{R}_{\phi_n}$ of length n, for ϕ_n from Lemma 6.2.2.*

2. *The remaining bits of x (called the proof part of x) is a string $\pi(y)$ of length $p(n)$, where $\pi(y) \in \{0,1\}^{p(n)}$ is a proof such that the algorithm V (from Claim 6.2.6) accepts when given oracle access to y and $\pi(y)$.*

For simplicity, in the above definition, we restrict our attention to $n \in \aleph$ such that n divides $p(n)\log n$ exactly.

6.2.2 Proof of Separation

In this section, we prove Theorem 6.2.1. Lemmas 6.2.8 and 6.2.9 prove the first and second parts of Theorem 6.2.1, respectively.

To test for $\mathcal{P}$, our tester (Algorithm 6.1), given oracle access to a string x, first checks, via random sampling, that the input part of x is equal to a repetition of the same string y. It then checks whether this y satisfies the hard CNF property (Lemma 6.2.2) by running the PCPP verifier guaranteed by Claim 6.2.6 with oracle access to y and the proof part of x. If x is far from $\mathcal{P}$, then, either the input part of x is far from repetitions of the same string and the first step rejects. Otherwise, the string y such that the input part of x is repetitions of y, is far from the hard CNF property. In this case, the second step rejects.

Showing that erasure-resilient testing of $\mathcal{P}$ is hard is achieved via a straightforward reduction from the problem of testing the hard CNF property without access to a proof. In particular, if one can efficiently test $\mathcal{P}$ with the proof part fully erased, one can also efficiently test the hard CNF property in the standard model without access to a proof, which is a contradiction to Lemma 6.2.2.

The following lemma shows that it is easy to test $\mathcal{P}$ in the standard model.

Lemma 6.2.8. *Let $\varepsilon \in (0,1)$. The property $\mathcal{P}$ can be ε-tested in the standard property testing model using $O(1/\varepsilon)$ queries.*

Proof. Algorithm 6.1 is an ε-tester for $\mathcal{P}$. Consider a string $x \in \{0,1\}^N$. Assume

Algorithm 6.1 Tester for $\mathcal{P}$

```
Input: $\varepsilon \in (0,1)$, $n \in \aleph$; oracle access to $x \in \{0,1\}^N$, where $N = (\log n + 1) \cdot p(n)$
    ▷ Set $s \leftarrow \frac{p(n)\log n}{n}$.
1: repeat $\lceil 4/\varepsilon \rceil$ times:
2:     Sample $i \in [n]$ and $j \in [2, s] \cap \mathbb{N}$ uniformly at random.
3:     Reject if $x[i] \neq x[(j-1)n + i]$.
4: Run $V$, from Claim 6.2.6, with parameter $\varepsilon/3$ and oracle access to the concatenation of $x[1 \dots n]$ and $x[p(n)+1 \dots 2 \cdot p(n)]$.
5: Reject if $V$ rejects; otherwise accept.
```

that satisfies $\mathcal{P}$. Then, there exists a string $y \in \{0,1\}^n$ such that $x[1 \dots p(n)]$ (the input part of x) is a repetition of y and the proof part of x is a proof π such that the verifier V from Claim 6.2.6 accepts when given oracle access to $y \circ \pi$. Therefore, Algorithm 6.1 accepts x.

Assume that x is ε-far from $\mathcal{P}$. Let $y' \in \{0,1\}^n$ denote the string $x[1 \dots n]$. If the input part of x is $\varepsilon/2$-far from repetitions of y', Steps 1-3 reject with probability at least $2/3$ (and we are done). If the input part of x is $\varepsilon/2$-close to repetitions of y', then we argue that y' has to be $\varepsilon/3$-far from satisfying the hard CNF property $\mathcal{R}_{\phi_n}$. If not, we can change x to a string satisfying $\mathcal{P}$ by (1) first modifying at most $\frac{\varepsilon}{2} \cdot p(n) \log n$ points in the plain part to make the plain part repetitions of y', (2) then changing the plain part to the repetitions of a string y'' satisfying $\mathcal{R}_{\phi_n}$ by modifying at most $\frac{\varepsilon}{2} \cdot p(n) \log n$ points, and (3) finally, changing at most $p(n)$ points in the proof part (all of it) to make it into a string π such that π is a proof that makes the verifier V from Claim 6.2.6 accept y''. In this process, we have modified strictly less than εN points in x, which contradicts our assumption that x is ε-far from $\mathcal{P}$. Since y' is $\varepsilon/3$-far from $\mathcal{R}_{\phi_n}$, by Claim 6.2.6 the verifier V, with probability at least $2/3$, rejects in Step 4. □

The following lemma, together with Lemma 6.2.8, implies Theorem 6.2.1.

Lemma 6.2.9. *Let $\varepsilon^\star \in (0,1)$ be the constant from Lemma 6.2.2. For all constants $\alpha \in (0,1)$ and $\varepsilon \in (0, \frac{\varepsilon^\star}{2}]$ satisfying $\alpha + \varepsilon < 1$, every α-erasure-resilient ε-tester for $\mathcal{P}$ makes $\tilde{\Omega}(N)$ queries, where N is the size of the input.*

Proof. The proof is by a simple reduction from $\varepsilon^\star$-testing of $\mathcal{R}_{\phi_n}$ in the standard model, which, by Lemma 6.2.2, has query complexity $\Omega(n)$.

Given a oracle access to a string $y \in \{0,1\}^n$ for which we need to $\varepsilon^\star$-test $\mathcal{R}_{\pi_n}$, we can simulate oracle access to a partially erased string $x \in \{0,1,\perp\}^N$, where $N = (\log n + 1) \cdot p(n)$ and $p(n)$ is as in Definition 6.2.7. The first $p(n) \log n$ bits of x are repetitions of y. The remaining $p(n)$ bits of x are equal to $\perp$.

A single query to x can be simulated by at most one query to y.

If $y \in \mathcal{R}_{\pi_n}$, then x satisfies $\mathcal{P}$, since we can complete the proof part of x to be a valid proof π that makes the verifier V from Claim 6.2.6 accept y.

If y is $\varepsilon^\star$-far from satisfying $\mathcal{R}_{\pi_n}$, then x is $\frac{\log n}{\log n+1} \cdot \varepsilon^\star$-far from $\mathcal{P}$ since each n-length block among the first $p(n) \log n$ bits of x is $\varepsilon^\star$-far from $\mathcal{R}_{\pi_n}$. Note that $\frac{\log n}{\log n+1} \geq \frac{1}{2}$ for large n. The fraction of erasures in x is $1/\log n$, which is smaller than every constant α. Therefore, an α-erasure resilient $\frac{\varepsilon^\star}{2}$-tester for $\mathcal{P}$ for constant α would yield a $\varepsilon^\star$-tester for $\mathcal{R}_{\pi_n}$ with the same query complexity. Since every $\varepsilon^\star$-tester for $\mathcal{R}_{\pi_n}$ requires $\Omega(n)$ queries on inputs of length n, every α-erasure-resilient $\frac{\varepsilon^\star}{2}$-tester for $\mathcal{P}$ requires $\Omega(n)$ queries. As $N = (\log n + 1) \cdot p(n)$, this implies that every α-erasure-resilient $\frac{\varepsilon^\star}{2}$-tester for $\mathcal{P}$ requires $\tilde{\Omega}(N)$ queries. $\square$

6.3 SEPARATING ERASURE-RESILIENT AND TOLERANT TESTING

6.3.1 Hadamard-based Separation

In this section, we describe a property $\mathcal{P}'$ that is erasure-resiliently testable using a constant number of queries, but not tolerantly testable using a constant number of queries, and prove Theorem 6.1.2. In fact, we prove the following (more general) statement and show that it implies Theorem 6.1.2.

Theorem 6.3.1. *Let $\varepsilon^\star \in (0, \frac{1}{100})$ be a constant. There exists a property $\mathcal{P} \subseteq \{0,1\}^*$ such that*

- *for every $\alpha \in [0, \frac{3\varepsilon^\star}{16})$ and $\varepsilon \in (\frac{3\varepsilon^\star}{4}, 1)$ such that $\alpha + \varepsilon < 1$, the property $\mathcal{P}'$ can be α-erasure-resiliently ε-tested using $O(\frac{1}{\varepsilon})$ queries.*
- *for all $\alpha \in (\frac{\varepsilon^\star}{8}, 1)$ and $\varepsilon' \in (\alpha, \varepsilon^\star - \frac{(\varepsilon^\star)^2}{4})$, the query complexity of (α, ε')-tolerant testing $\mathcal{P}'$ on inputs of length N is $\tilde{\Omega}(\log N)$.*

6.3.2 Separating Property $\mathcal{P}'$: PCPs of Proximity and Hadamard Code

Just as in the case of the property $\mathcal{P}$ (Definition 6.2.7), the property $\mathcal{P}'$ is also defined in terms of a property $\mathcal{R}$ that is hard to test in the standard property testing model (Goldreich et al. (1998); Rubinfeld & Sudan (1996)), a PCPP (Ergün et al. (2004); Ben-Sasson et al. (2006); Dinur & Reingold (2006)) for the problem of testing $\mathcal{R}$. The additional ingredient in the definition here is the Hadamard code. The idea of using PCPs of proximity in separating the two property testing models comes from the work of Fischer & Fortnow (2006). Our contribution is to use locally list decodable codes in this context.

The following is the definition of our separating property $\mathcal{P}'$. At a high level, the definition says that, for all $n \in \aleph$, a string of length $O(2^{n \cdot \text{polylog } n})$ satisfies $\mathcal{P}'$ if its first part is the repetition of a string y satisfying $\mathcal{R}_{\phi_n}$, and the second part is the encoding (by the Hadamard code) of y concatenated with a proof π that makes the algorithm V in Claim 6.2.6 accept.

Definition 6.3.2 (Separating Property $\mathcal{P}'$)**.** *Let $\varepsilon^\star \in (0,1)$ and $\aleph \subseteq \mathbb{N}$ be as in Lemma 6.2.2. For $n \in \aleph$, let $p(n) \le cn \cdot \text{polylog } n$ denote the length of proof that the algorithm V in Claim 6.2.6 has oracle access to. A string $x \in \{0,1\}^N$ of length $N = \frac{4}{\varepsilon^\star} \cdot 2^{n+p(n)}$ satisfies $\mathcal{P}'$ if the following conditions hold:*

1. *The first $(\frac{4}{\varepsilon^\star} - 1) \cdot 2^{n+p(n)}$ bits of x (called the plain part of x) consist of $(\frac{4}{\varepsilon^\star} - 1) \cdot \frac{2^{n+p(n)}}{n}$ repetitions of a string $y \in \mathcal{R}_{\phi_n}$ of length n, for ϕ_n from Lemma 6.2.2.*

2. *The remaining bits of x (called the encoded part of x) form the Hadamard encoding of a string $y \circ \pi(y)$ of length $n+p(n)$, where $\circ$ denotes the concatenation operation on strings. The string $y \in \{0,1\}^n$ is the same as the one in the description of the plain part. The string $\pi(y) \in \{0,1\}^{p(n)}$ is a proof such that the algorithm V (from Claim 6.2.6) accepts when given oracle access to y and $\pi(y)$.*

6.3.3 Proof of Hadamard-based Separation

In this section, we prove Theorem 6.3.1, which in turn implies Theorem 6.1.2. Lemmas 6.3.3 and 6.3.6 prove the first and second parts of Theorem 6.3.1, respectively.

We first give a high level overview of the proof. The erasure-resilient tester for $\mathcal{P}'$ first obtains a list of (implicit) decodings of the encoded part (see Definition 6.3.2) of an input string $x \in \{0,1\}^N$ using the local list erasure-decoder guaranteed by Theorem 5.1.10. If $x \in \mathcal{P}$, with high probability, at least one of the algorithms implicitly computes (see Definition 5.1.2) the string $y \circ \pi(y)$, where y is such that the plain part of x (see Definition 6.3.2) consists of repetitions of y, and $\pi(y)$ is a proof string such that the algorithm V (from Claim 6.2.6) accepts upon oracle access to $y \circ \pi(y)$. In case x is ε-far from $\mathcal{P}'$ we show that for every algorithm T output by the local list erasure-decoder, the string $y' \circ \pi(y')$ implicitly computed by T is such that, (1) either the plain part of x is far from repetitions of y', (2) or y' is far from $\mathcal{R}$ (in which case, the algorithm V from Claim 6.2.6 rejects when given oracle access to $y' \circ \pi(y')$).

To show that tolerant testing of $\mathcal{P}'$ is hard, we reduce $\varepsilon^\star$-testing of $\mathcal{R}_{\phi_n}$ to it. Specifically, given oracle access to a string $y \in \{0,1\}^n$ that we want to $\varepsilon^\star$-test, we simulate oracle access to a string $x \in \{0,1\}^N$ such that the plain part of x consists of repetitions of y, and every bit in the encoded part of x is 0. Since every Hadamard codeword has an equal number of 0s and 1s, the string x can be thought of as having 0.5 fraction of "errors" in the encoded part. If $y \in \mathcal{R}_{\phi_n}$, then the string x is close to $\mathcal{P}'$, as the errors are only in the encoded part of x and the length of the encoded part is a small fraction of the length of x. If y is far from $\mathcal{R}_{\phi_n}$, then x is also far from $\mathcal{P}'$, since the plain part of x, whose length is a large fraction of the length of x, is the repetitions of y. Thus, the decision of a tolerant tester for $\mathcal{P}'$ on x can be used to test y for $\mathcal{R}_{\phi_n}$, implying that the complexity of tolerant testing of $\mathcal{P}'$ is equal to the complexity of testing $\mathcal{R}_{\phi_n}$.

We now prove the existence of an efficient erasure-resilient tester for $\mathcal{P}'$. An α-erased string x is ε-far from a property $\mathcal{P}'$ if there is no way to complete x to a string that satisfies $\mathcal{P}'$ without changing at least $\varepsilon \cdot |x|$ nonerased values in x.

Lemma 6.3.3. *Let $\varepsilon^\star \in (0,1)$ be as in Lemma 6.2.2. For every $\alpha \in [0, \frac{3\varepsilon^\star}{16})$ and $\varepsilon \in (\frac{3\varepsilon^\star}{4}, 1)$ such that $\alpha + \varepsilon < 1$, the property $\mathcal{P}'$ can be α-erasure-resiliently ε-tested* *using $O(\frac{1}{\varepsilon})$ queries.*

Proof. The erasure-resilient tester for $\mathcal{P}'$ is described in Algorithm 6.2. The query complexity of the tester is evident from its description. We now prove that the tester, with probability at least $\frac{2}{3}$, accepts strings in $\mathcal{P}'$ and rejects strings that are ε-far from $\mathcal{P}'$.

Algorithm 6.2 Erasure-resilient tester for separating property $\mathcal{P}'$

```
Input: $\alpha, \varepsilon \in (0,1), N = \frac{4}{\varepsilon^\star} \cdot 2^{(n+p(n))}$; oracle access to $x \in \{0,1,\perp\}^N$
    ▷ Set $s \leftarrow (\frac{4}{\varepsilon^\star} - 1) \cdot \frac{2^{(n+p(n))}}{n}$, $\varepsilon' \leftarrow \frac{\varepsilon}{3}$.
    ▷ Set $Q \leftarrow \frac{C}{\varepsilon}$ for a large enough constant $C$.
 1: Accept whenever the number of queries exceeds $Q$.
 2: Run a $(\frac{3}{4}, q, L)$-local list erasure-decoder for the Hadamard code (Algorithm 5.1)
    with oracle access to $x[sn+1..N]$, the encoded part of $x$.
    ▷ Let $T_1, T_2, \ldots, T_L$ be the list of algorithms returned in the above step.
 3: for each $k \in [L]$ do
    ▷ Check if the plain part of $x$ is the repetition of $y$, where $y$ denotes the first $n$ bits
    of the decoding (given by $T_k$) of the encoded part of $x$.
 4:     repeat $\lceil \frac{9 \log L}{\varepsilon} \rceil$ times:
 5:         Pick $a \in_R [n], i \in_R [s]$.
 6:         if $x[(i-1)n+a] \neq \perp$ and $T_k(a) \neq x[(i-1)n+a]$ then
 7:             Discard the current $k$
    ▷ Check if the string $y \in \mathcal{R}_{\phi_n}$, where $y$ denotes the first $n$ bits of the decoding (by
    $T_k$) of the encoded part of $x$.
 8:     repeat $\lceil 4 \log L \rceil$ times:
 9:         Run $V$, from Claim 6.2.6, with input $\varepsilon'$ and oracle access to $T_k$.
10:         Discard the current $k$ if $V$ rejects.
11: Reject if every $k \in [L]$ is discarded; otherwise, accept.
```

Let $\aleph, \varepsilon^\star \in (0,1)$ be as in Lemma 6.2.2. Fix $n \in \aleph$ and let $p(n)$ and N be as in Definition 6.3.2. Let s denote $(\frac{4}{\varepsilon^\star} - 1) \cdot \frac{2^{n+p(n)}}{n}$. Consider a string $x \in \{0,1\}^N$ that we want to erasure-resiliently test for $\mathcal{P}'$. As in Definition 6.3.2, we refer to the substring $x[1 \ldots sn]$ as the plain part of x and the substring $x[sn+1 \ldots N]$ as the encoded part of x.

Assume that $x \in \mathcal{P}$. By this assumption, we can see that there exists a string

$y \circ \pi \in \{0,1\}^{n+p(n)}$ such that (1) $y \in \mathcal{R}_{\phi_n}$ and the plain part of x can be completed to a repetition of y, (2) π is a proof such that the algorithm V (from Claim 6.2.6) accepts when given oracle access to $y \circ \pi$, and (3) the encoded part of x can be completed to the Hadamard encoding of $y \circ \pi$. Since $\alpha < 3\varepsilon^\star/16$, the fraction of erasures in the encoded part of x is less than $3/4$. Hence, by Theorem 5.1.10, with probability at least $2/3$, there exists an algorithm T_k computed in Step 2 of Algorithm 6.2 such that T_k implicitly computes the string $y \circ \pi \in \{0,1\}^{n+p(n)}$. Therefore k is not discarded in either Step 7 or Step 10. Thus, the tester will accept with probability at least $2/3$.

Now, assume that x is ε-far from $\mathcal{P}'$. Let E denote the event that the number of queries made by the tester does not exceed its query budget. We first show that, conditioned on E, the tester rejects with probability at least $4/5$.

Claim 6.3.4. *The plain part of x is $\frac{2\varepsilon}{3}$-far from s repetitions of a string $y \in \mathcal{R}_{\phi_n}$.*

Proof. Since x is ε-far from satisfying $\mathcal{P}'$, at least εN nonerased values in x need to be changed in order to complete it to a string satisfying $\mathcal{P}'$. The upper bound on the number of nonerased values in the encoded part of x is $\frac{\varepsilon^\star}{4} \cdot N$, which is at most $\varepsilon N/3$ since $\varepsilon \in (\frac{3\varepsilon^\star}{4}, 1)$. Thus, the plain part of x needs to be changed in at least $2\varepsilon N/3$ values in order for it to be s repetitions of a string $y \in \mathcal{R}_{\phi_n}$. The claim follows. □

From Claim 6.3.4, it follows that at least $\frac{2\varepsilon \cdot sn}{3}$ nonerased points need to be changed in the plain part of x for it to be s repetitions of a string $y \in \mathcal{R}_{\phi_n}$.

Claim 6.3.5. *For every $y \in \{0,1\}^n$, if the plain part of x can be changed to s repetitions of y by modifying less than $\frac{\varepsilon \cdot sn}{3}$ nonerased values, then y is $\frac{\varepsilon}{3}$-far from $\mathcal{R}_{\phi_n}$.*

Proof. Consider $y \in \{0,1\}^n$ such that we can change less than $\varepsilon \cdot sn/3$ nonerased points in the plain part of x and make it s repetitions of y. Assume that there exists $y' \in \mathcal{R}_{\phi_n}$ such that the Hamming distance of y' to y is at most $\varepsilon \cdot n/3$. Then, the plain part of x, can be changed to be s repetitions of y' by first changing it to be s repetitions of y (modifying less than $\varepsilon \cdot sn/3$ nonerased points) and then modifying at most $s \cdot \varepsilon \cdot n/3$ nonerased points to make it s repetitions of y'. In other words, $x[1 \ldots sn]$ can be modified in less than $2\varepsilon \cdot sn/3$ nonerased points to make it s repetitions of a string y' in $\mathcal{R}_{\phi_n}$. This contradicts Claim 6.3.4. □

Fix $k \in [L]$, where L is the number of algorithms returned by the local list erasure-decoder. Let $y' \in \{0,1\}^n$ be the first n bits from the left in the decoding, using T_k, of the encoded part of x. We will show that the algorithm discards k with high probability. We split the analysis into two cases.

Case I: Suppose we need to change at least $\frac{\varepsilon \cdot sn}{3}$ nonerased points in the plain part of x for it to become s repetitions of y'. We show that in this case, Steps 4-7 discard k with probability at least $\frac{9}{10L}$. A point $(i-1)n + a$ for $i \in [s]$ and $a \in [n]$ is called a witness if $x[(i-1)n + a] \neq \perp$ and $x[(i-1)n + a] \neq y'[a]$. Since we need to change at least $\varepsilon \cdot sn/3$ nonerased points in the plain part of x for it to become s repetitions of y', there are at least $\varepsilon \cdot sn/3$ witnesses in the plain part of x. In each iteration of Steps 4-7, the point selected is a witness with probability at least $\frac{\varepsilon \cdot sn}{3sn} = \frac{\varepsilon}{3}$. Thus, in $\lceil \frac{9 \log L}{\varepsilon} \rceil$ iterations, Algorithm 6.2 finds a witness (and discards k) with probability at least $9/10L$.

Case II: In this case, we assume that we can change less than $\varepsilon \cdot sn/3$ nonerased points in the plain part of x and make it s repetitions of y'. Then, by Claim 6.3.5, y' is $\varepsilon/3$-far from $\mathcal{R}_{\phi_n}$. Let $\varepsilon' = \frac{\varepsilon}{3}$. By Claim 6.2.6, for every proof $\pi \in \{0,1\}^{p(n)}$, the algorithm V (from Claim 6.2.6), on input ε' and oracle access to $y' \circ \pi$ (obtained via T_k), rejects (causing k to be discarded) with probability at least $2/3$. Thus, the probability that tester fails to discard k in $\lceil 4 \log L \rceil$ independent iterations of Steps 8-10 is at most $1/16L$.

Therefore, the probability that the tester fails to discard k is at most $\frac{1}{10L} + \frac{1}{16L} <$

$\frac{1}{5L}$. By the union bound, the probability that Algorithm 6.2 fails to discard some $k \in [L]$ is at most $1/5$. Thus, conditioned on the event E that the number of queries made by the tester does not exceed its query budget, with probability at least $4/5$, the tester rejects.

We now bound the probability of the event E. For this, we calculate the expected number of queries made by Algorithm 6.2. For all $k \in [L]$, the expected number of queries that each invocation of the algorithm T_k makes is at most q. Since the fraction of erasures (with respect to the encoded part $x[sn+1 \dots N]$) is at most $3/4$, the values q and L are both constants (by Theorem 5.1.10). Hence, the expected number of queries made in Steps 4-7 is $O(\frac{1}{\varepsilon})$.

By Claim 6.2.6, the number of queries made by the algorithm V (from Claim 6.2.6) on input $\varepsilon' = \frac{\varepsilon}{3}$ and oracle access to T_k, is $O(\frac{1}{\varepsilon})$. Thus, the expected number of queries made in Steps 8-10 by Algorithm 6.2 is $O(\frac{1}{\varepsilon})$.

Therefore the expected total number of queries made by the tester is $O(\frac{1}{\varepsilon})$. Hence, for a large enough constant C, the probability that the number of queries exceed $\frac{C}{\varepsilon}$ is at most $1/10$ by Markov's inequality. Hence, the probability that the tester accepts x that is ε-far from $\mathcal{P}'$ is at most $1/3$. $\square$

Lemma 6.3.6. *Let $\varepsilon^\star \in (0,1)$ be as in Lemma 6.2.2. For every $\alpha \in (\frac{\varepsilon^\star}{8}, 1)$ and $\varepsilon' \in (\alpha, \varepsilon^\star - \frac{(\varepsilon^\star)^2}{4})$, the query complexity of (α, ε')-tolerant testing $\mathcal{P}'$ on strings of length N is $\tilde{\Omega}(\log N)$.*

Proof. Let $\aleph, \varepsilon^\star \in (0,1)$ be as in Lemma 6.2.2. We will prove the lemma by showing a reduction from $\varepsilon^\star$-testing of $\mathcal{R}_{\phi_n}$. Fix $n \in \aleph$ and let $p(n)$ and N be as in Definition 6.3.2. Let s denote $(\frac{4}{\varepsilon^\star} - 1) \cdot \frac{2^{n+p(n)}}{n}$.

Consider a string $y \in \{0,1\}^n$ that we want to $\varepsilon^\star$-test for $\mathcal{R}_{\phi_n}$. Let $x \in \{0,1\}^N$ be the string where the first sn bits of x are s repetitions of y and the remaining bits are all 0s. We refer to the substring $x[1 \dots sn]$ as the plain part of x and the substring

$x[sn+1\dots N]$ as the encoded part of x.

Assume that A is an (α,ε')-tolerant tester for $\mathcal{P}'$. We now describe an $\varepsilon^\star$-tester A' for $\mathcal{R}_{\phi_n}$ that has the same query complexity as A. Given oracle access to $y \in \{0,1\}^n$, the tester A' runs the tester A on the string $x \in \{0,1\}^N$ and accepts if and only if A accepts, where x is constructed from y as described above. Observe that one can simulate a query to x by making at most one query to y.

If $y \in \mathcal{R}_{\phi_n}$, then x is α-close to $\mathcal{P}'$. Observe that the encoded part of x needs to be changed in at most $1/2$ fraction of its positions in order to make it the encoding of a string $y \circ \pi$, where π is a proof that makes a PCP of proximity for testing $\mathcal{R}_{\phi_n}$ accept. This follows from the fact that the normalized weight of every nonzero codeword in the Hadamard code is $1/2$. Thus, the fraction of bits in x that needs to be changed in order to make it satisfy $\mathcal{P}'$ is at most $\frac{1}{2}\cdot\frac{N-sn}{N}=\frac{\varepsilon^\star}{8}$, which is less than α. Therefore, by definition, A' will accept x with probability at least $2/3$.

Assume now that y is $\varepsilon^\star$-far from $\mathcal{R}_{\phi_n}$. Then x needs to be changed in at least $\varepsilon^\star \cdot sn$ positions to make it satisfy $\mathcal{P}'$. From this, one can observe that x is $(\varepsilon^\star - \frac{(\varepsilon^\star)^2}{4})$-far from $\mathcal{P}'$. Hence, for all $\varepsilon' < \varepsilon^\star - \frac{(\varepsilon^\star)^2}{4}$, we have that A will reject x with probability at least $2/3$, and therefore A' will reject y with probability at least $2/3$.

Thus, we have shown that the query complexity of (α,ε')-tolerant testing $\mathcal{P}'$ is at least the query complexity of $\varepsilon^\star$-testing $\mathcal{R}_{\phi_n}$. Hence, the query complexity of (α,ε')-tolerant testing $\mathcal{P}'$ is $\Omega(n)$, which is equal to $\tilde{\Omega}(\log N)$. □

Proof of Theorem 6.1.2. Theorem 6.3.1 states that, for certain ranges of parameters $\alpha,\varepsilon,\varepsilon' \in (0,1)$ and for large enough $N \in \mathbb{N}$, the property $\mathcal{P}'$ on binary strings of length N, is α-erasure-resiliently ε-testable, but is not (α,ε')-tolerantly testable. To prove Theorem 6.1.2, we need to show the existence of $\alpha,\varepsilon \in (0,1)$ such that the property $\mathcal{P}'$ on binary strings of length N is α-erasure-resiliently ε-testable, but is not $(\alpha,\alpha+\varepsilon)$-tolerantly testable. In other words, the following system of inequalities should have a solution.

$$\frac{\varepsilon^\star}{8} < \alpha < \frac{3\varepsilon^\star}{16}; \qquad \frac{3\varepsilon^\star}{4} < \varepsilon < 1;$$

$$\varepsilon' = \alpha + \varepsilon < \varepsilon^\star - (\varepsilon^\star)^2/4$$

For every $0 < \varepsilon^\star < 1/100$, the value $\varepsilon^\star - (\varepsilon^\star)^2/4$ is strictly greater than $\varepsilon^\star - \varepsilon^\star/400 = 399\varepsilon^\star/400$. For $\alpha = \varepsilon^\star/6$ and $\varepsilon = 4\varepsilon^\star/5$, which satisfy the first two inequalities, we can see that $\alpha + \varepsilon = 29\varepsilon^\star/30 < 399\varepsilon^\star/400 < \varepsilon^\star - (\varepsilon^\star)^2/4$. Thus there exists $\alpha, \varepsilon \in (0,1)$ satisfying $\alpha + \varepsilon < 1$ such that $\mathcal{P}'$ is α-erasure-resiliently ε-testable, but not $(\alpha, \alpha+\varepsilon)$-tolerantly testable. Theorem 6.1.2 follows. $\square$

6.3.4 Strengthened Separation

In this section, we describe a property $\mathcal{P}''$ that can be erasure-resiliently tested using a constant number of queries, but for which every tolerant tester has query complexity $n^{\Omega(1)}$, and prove Theorem 6.1.3. The following theorem implies Theorem 6.1.3.

Theorem 6.3.7. *There exists a property $\mathcal{P}''$ and constants $\varepsilon^\star \in (0,1), c_2 > 1$ such that,*

- *For every $\varepsilon \in \left(\frac{\varepsilon^\star}{8}, 1\right)$ and $\alpha \in (0, \frac{\varepsilon^\star}{57600 \cdot c_2})$ such that $\alpha + \varepsilon < 1$, property $\mathcal{P}''$ can be α-erasure-resiliently ε-tested using $O(\frac{1}{\varepsilon})$ queries,*
- *For every $\alpha \in (\frac{\varepsilon^\star}{57600 \cdot c_2 + 2\varepsilon^\star}, 1)$, and $\varepsilon' \in \left(\alpha, \frac{28800 \cdot c_2 \cdot \varepsilon^\star}{28800 \cdot c_2 + \varepsilon^\star}\right)$, every (α, ε')-tolerant tester for $\mathcal{P}''$ on inputs of length N has query complexity $N^{\Omega(1)}$.*

6.3.5 Separating Property $\mathcal{P}''$: Smooth PCPs of Proximity and Approximate Locally List Erasure-Decodable Codes

The property $\mathcal{P}''$ is very similar to the property $\mathcal{P}'$ that we used in our Hadamard-based separation (see Definition 6.3.2). Like a string that satisfies $\mathcal{P}'$, a string that satisfies $\mathcal{P}''$ can also be thought of as consisting of a plain part (that contains the repetition of a string $y \in \mathcal{R}_{\phi_n}$) and an encoded part. The encoded part of a string in $\mathcal{P}'$ is the Hadamard encoding of a string $y \circ \pi$, where π is a proof that makes the algorithm V from Claim 6.2.6 accept. However, the encoded part of a string satisfying $\mathcal{P}''$ is the encoding of a string π', where π' is a proof (whose length is asymptotically equal to $|\pi|$) that makes a 'smooth' PCPP accept. In addition, the encoding uses an ALLEDC (see Definition 5.1.4) instead of the Hadamard code.

We first describe the 'smooth' PCPP used in our construction. The following lemma by Ben-Sasson et al. (2006) and (Guruswami & Rudra, 2005, Lemma 5) states that algorithms making nonadaptive queries can be transformed into algorithms that make nearly uniform queries.

Lemma 6.3.8 (Guruswami & Rudra (2005); Ben-Sasson et al. (2006))**.** *Let $n \in \mathbb{N}$. Consider a nonadaptive algorithm T that gets oracle access to strings from $\{0,1\}^n$. There exists a mapping $\varphi_T : \{0,1\}^n \to \{0,1\}^{n'}$ and an algorithm T' satisfying the following:*

- *For every $x \in \{0,1\}^n$, the distribution on outcomes of T with oracle access to x is identical to the distribution on outcomes of T' with oracle access to $\varphi_T(x)$. Moreover, $3n < n' \le 4n$, and the number of queries that T' makes to $\varphi_T(x)$ is at most twice the number of queries that T makes to x.*
- *Upon oracle access to $x' \in \{0,1\}^{n'}$, each query of T' is to location $j \in [n']$ with probability at most $2/n'$.*

Combining Lemma 6.2.4 with Lemma 6.3.8 (along with the fact that $\mathcal{R}$ can be decided using linear-sized circuits), we get the required 'smooth' PCPP for $\mathcal{R}$.

Lemma 6.3.9 (Smooth PCPP)**.** *Let $c_1 > 0, c_2 > 1$ be fixed constants. Let $n \in \mathbb{N}$. The property $\mathcal{R}_{\phi_n}$ has a PCPP V that works for all $\varepsilon \in (0, 1]$, gets oracle access to an input y of length n and a proof π of length at most $c_1 n \cdot \operatorname{poly}\log n$, and makes at most $\frac{c_2}{\varepsilon}$ queries. Moreover, the queries of V are nonadaptive and satisfy the following:*

- *each query V makes to y is to any particular location of y with probability $1/n$;*
- *each query V makes to π is to any particular location of π with probability at most $2/|\pi|$.*

Proof. Let $c > 0$ be the constant from Claim 6.2.6. Consider the algorithm V' guaranteed by Lemma 6.2.4 for the property $\mathcal{R}_{\phi_n}$. The algorithm V' gets oracle access to the concatenation of an input $y \in \{0,1\}^n$ and a proof $\pi' \in \{0,1\}^{p'(n)}$, where $p'(n) \le cn \cdot \operatorname{poly}\log n$.

We now describe an algorithm V'' that, on oracle access to a string $y \circ \pi''$, where $y \in \{0,1\}^n$ and $\pi'' \in \{0,1\}^{n+p'(n)}$, and does the following:

1. Sample a uniformly random $i \in [n]$ and **reject** if $y[i] \ne \pi''[i]$.
2. Simulate V' with oracle access to π'' and **reject** if V' rejects.
3. **Accept** if neither of the above events happen.

We prove the following claim about the algorithm V''.

Claim 6.3.10. *V'' is an algorithm satisfying:*

- *if $y \in \mathcal{R}_{\phi_n}$, then for some proof π'', the algorithm V' always accepts $y \circ \pi''$;*
- *if $y \notin \mathcal{R}_{\phi_n}$, then for every π'', the algorithm V' rejects $y \circ \pi''$ with probability proportional to the relative Hamming distance of y from $\mathcal{R}_{\phi_n}$.*

Proof. Assume $y \in \mathcal{R}_{\phi_n}$. There exists a proof π' of length at most $cn \cdot \operatorname{poly}\log n$ such that the algorithm V' accepts when given oracle access to $y \circ \pi'$. Therefore, algorithm V'' accepts if given oracle access to $y \circ \pi''$, where $\pi'' = y \circ \pi'$.

Next, assume that $y \notin \mathcal{R}_{\phi_n}$. Let δ be the relative Hamming distance of y from $\mathcal{R}_{\phi_n}$. Fix $\pi'' \in \{0,1\}^{n+p'(n)}$. Let δ' be the relative Hamming distance of y from the string y' obtained by considering the first n bits of π''. Step 1 of the algorithm V'' rejects with probability δ', since, for a uniformly random index $i \in [n]$, we have that $y[i] \neq y'[i]$ with probability δ'. If $\delta' \geq \delta/2$, then Step 1 of algorithm V'' rejects with probability at least $\delta/2$. If $\delta' < \delta/2$, then the relative Hamming distance of y' from $\mathcal{R}_{\phi_n}$ has to be greater than $\delta/2$; otherwise, the distance of y from $\mathcal{R}_{\phi_n}$ is less than δ, which is a contradiction. If y' has distance at least $\delta/2$ from $\mathcal{R}_{\phi_n}$, for every string $z \in \{0,1\}^{p'(n)}$ that forms the last $p'(n)$ bits of π'', the algorithm V' with oracle access to $\pi'' = y' \circ z$ rejects with probability $\Omega(\delta)$. That is, Step 2 of V'' rejects with probability $\Omega(\delta)$. □

We can think of V'' as running two algorithms V_1 and V_2, where V_1 makes the input queries of V'' and V_2 makes the proof queries of V''. We observe that the query distribution of V_1 is uniform over the input part. By applying Lemma 6.3.8 to V_2 we obtain a mapping $\varphi : \{0,1\}^* \to \{0,1\}^*$ and an algorithm V_2' such that each query of V_2' is to a particular location in the string $\varphi(\pi'')$ with probability at most $2/|\varphi(\pi'')|$. By Lemma 6.3.8, we also have: $|\varphi(\pi'')| \leq 4|\pi''|$.

Let $p(n)$ denote $|\varphi(\pi'')|$, where $\pi'' \in \{0,1\}^{n+p'(n)}$. Consider the algorithm V''' that runs V_1 and V_2' using a common random string with oracle access to a string $y \circ z$, where $y \in \{0,1\}^n$ and $z \in \{0,1\}^{p(n)}$, and rejecting whenever V'' rejects based on the query answers. In addition, V''' also rejects if it detects any evidence that z is not in the image of φ.

If $y \in \mathcal{R}_{\phi_n}$, then there exists a proof π'' such that V'' accepts $y \circ \pi''$, implying that for the same π'', the algorithm V''' accepts $y \circ \pi$, where $\pi = \varphi(\pi'')$. If $y \notin \mathcal{R}_{\phi_n}$, then for every proof π'', the algorithm V'' rejects $y \circ \pi''$ with probability proportional to the relative Hamming distance of y from $\mathcal{R}_{\phi_n}$. This implies that for every proof π, the algorithm V''' rejects $y \circ \pi$ with probability proportional to the relative Hamming distance of y from $\mathcal{R}_{\phi_n}$.

On input $\varepsilon \in (0,1)$, the algorithm V guaranteed by the statement of the lemma repeats for $\Theta(1/\varepsilon)$ time, the algorithm V'''. The acceptance and rejection guarantees of V are immediate. Note also that the distribution of a single input or proof query does not change by repetition. The lemma follows. $\square$

Next, we describe the approximate locally list decodable code that we use in our construction. We prove the existence of an approximate locally list erasure-decodable code (ALLEDC) with inverse polynomial rate. The following theorem due to Impagliazzo et al. (2010) proves the existence of an approximate locally decodable code with inverse polynomial rate, constant query complexity, and constant list size.

Theorem 6.3.11 (Impagliazzo et al. (2010) as restated by Ben-Aroya et al. (2010b))**.** *For every $\gamma, \beta > 0$, there exist a number $f(\gamma, \beta) > 0$ and a code family $\{C_k : \mathbb{F}_2^k \to \mathbb{F}_2^{f(\gamma,\beta)k^5}\}_{k\in\mathbb{N}}$ that is $(\gamma, \beta, O(\frac{\log(1/\beta)}{(\frac{1}{2}-\gamma)^3}), O(\frac{1}{(\frac{1}{2}-\gamma)^2}))$-approximate locally list decodable.*

To this code, we apply Observation 5.1.7 which states that every ALLDC that works in the presence of errors also works in the presence of twice as many erasures (with the same parameters up to constant factors). This gives us the required ALLEDC.

Lemma 6.3.12. *Let $c_3 > 0$ be a constant. For every $\gamma, \beta > 0$, there exist a number $f(\gamma, \beta) > 0$ and a code family $\{C_k : \mathbb{F}_2^k \to \mathbb{F}_2^{f(\gamma,\beta)k^5}\}_{k\in\mathbb{N}}$ that is $(\gamma, \beta, \frac{c_3 \log(1/\beta)}{(1-\gamma)^3}, \frac{c_3}{(1-\gamma)^2})$-approximate locally list erasure-decodable.*

The following is the definition of our separating property $\mathcal{P}''$. Note that the encoded part of a string satisfying $\mathcal{P}''$ contains the encoding of a proof as well as the complement of that encoding. This is done in order to equalize the number of 0s and 1s in the encoded part.

Definition 6.3.13 (Separating Property $\mathcal{P}''$). *Let $\aleph$, $\{\mathcal{R}_{\phi_n}\}_{n\in\aleph}$ and $\varepsilon^\star \in (0,1)$ be as in Lemma 6.2.2. Let $c_1 > 0$, $c_2 > 1$ be as in Lemma 6.3.9. Let $c_3 > 0$ be as in Lemma 6.3.12. Let $m = \frac{28800\cdot c_2}{\varepsilon^\star}$, $\gamma = \frac{1}{2} + \frac{\varepsilon^\star}{57600\cdot c_2}$ and $\beta = \frac{\varepsilon^\star}{9000c_2\cdot\left\lceil \ln\frac{6c_3}{(1-\gamma)^2}\right\rceil}$.*

For $n \in \aleph$, let $p(n) \le c_1 \cdot n \cdot \operatorname{polylog} n$ denote the length of a valid proof that makes the algorithm V from Lemma 6.3.9 accept. Let $f(\cdot,\cdot)$ be as in Lemma 6.3.12. Let $\mathcal{C} = \{C_k\}_{k\in\mathbb{N}}$ be the $(\gamma, \beta, \frac{c_3\log(1/\beta)}{(1-\gamma)^3}, \frac{c_3}{(1-\gamma)^2})$-ALLEDC from Lemma 6.3.12.

A string $x \in \{0,1\}^N$ of length $N = (m+1)\cdot 2f(\gamma,\beta)\cdot(p(n))^5$ satisfies $\mathcal{P}''$ if the following conditions hold:

1. *The first $m \cdot 2f(\gamma,\beta)\cdot(p(n))^5$ bits of x (called the plain part of x) consist of $m \cdot \frac{2f(\gamma,\beta)\cdot(p(n))^5}{n}$ repetitions of a string $y \in \{0,1\}^n$, where $y \in \mathcal{R}_{\phi_n}$ of length n.*

2. *The remaining $2f(\gamma,\beta)\cdot(p(n))^5$ bits of x is called the encoded part. Its first half is the encoding, using $\mathcal{C}$, of a string $\pi \in \{0,1\}^{p(n)}$ such that the PCPP V in Lemma 6.3.9 accepts when given oracle access to $y \circ \pi$. The second half of the encoded part is the complement of its first half.*

6.3.6 Proof of Strengthened Separation

In this section, we prove Theorem 6.3.7. Lemmas 6.3.14 and 6.3.18 together imply the first and second parts of Theorem 6.3.7, respectively. The high level idea of the proof of Lemma 6.3.14 is very similar to that of Lemma 6.3.3. The differences arise mainly because of the way the encoded parts of strings satisfying $\mathcal{P}'$ and $\mathcal{P}''$ differ. The erasure-resilient tester for $\mathcal{P}'$ could first check whether the plain part is a repetition of the 'decoded input', and then check whether the 'decoded input' is in $\mathcal{R}$ with the help of the 'decoded PCPP proof'. Since the encoded part of $\mathcal{P}''$ is the encoding of just a PCPP proof, this is not possible. Instead, the erasure-resilient tester for $\mathcal{P}''$ samples a uniformly random point u from the plain part and uses the 'block' from which u is obtained as a 'candidate input' y. It then checks whether the plain part is a repetition of y and also checks whether $y \in \mathcal{R}$ using the 'approximately decoded proof'. In case a string is α-erased and ε-far from $\mathcal{P}''$, we show that the 'candidate input' y that we sample is $c\alpha$-erased and $c'\varepsilon$-far from $\mathcal{R}$, for some constants c, c'. Hence, the smooth PCPP verifier rejects.

Lemma 6.3.14. *Let $\varepsilon^\star \in (0,1)$ be as in Lemma 6.2.2 and $c_2 > 1$ be as in Lemma 6.3.9. For every $\varepsilon \in \left(\frac{\varepsilon^\star}{8}, 1\right)$ and $\alpha \in (0, \frac{\varepsilon^\star}{57600 \cdot c_2})$ such that $\alpha + \varepsilon < 1$, the property $\mathcal{P}''$ is α-erasure-resiliently ε-testable using $O(\frac{1}{\varepsilon})$ queries.*

Proof. The erasure-resilient tester is presented in Algorithm 6.3. Let m denote $\frac{28800 \cdot c_2}{\varepsilon^\star}$. Let $\gamma = \frac{1}{2} + \frac{\varepsilon^\star}{57600 \cdot c_2}$, $\beta = \frac{\varepsilon^\star}{9000 c_2 \cdot \left\lceil \ln \frac{6c_3}{(1-\gamma)^2} \right\rceil}$, $q = \frac{c_3 \log(1/\beta)}{(1-\gamma)^3}$, and $L = \frac{c_3}{(1-\gamma)^2}$. For $n \in \aleph$, consider a string $x \in \{0,1\}^N$, where $N = (m+1) \cdot 2f(\gamma, \beta) \cdot (p(n))^5$. The plain part of x is m times larger than the encoded part. Let s denote the number $m \cdot \frac{2f(\gamma,\beta) \cdot (p(n))^5}{n}$.

Assume that x satisfies $\mathcal{P}''$. Since x satisfies $\mathcal{P}''$, the plain part of x is completable to the repetitions of y for some $y \in \mathcal{R}_{\phi_n}$. Therefore, Steps 2-7 never reject. By the definition of $\mathcal{P}''$, the first half of the encoded part of x is the encoding (using the (γ, β, q, L)-ALLED code $\mathcal{C}$ from Lemma 6.3.12) of a string $\pi(y) \in \{0,1\}^{p(n)}$ such that the smoothed PCPP V with oracle access to $y \circ \pi(y)$ always accepts. The second half of the encoding is completable to the complement of the first half. The fraction of erasures in the encoded part (even if all of the erasures were there) is at most $(m+1)\alpha$. Therefore, the fraction of erasures is at most $(m+1) \cdot \alpha = \frac{1}{2} + \frac{1}{2m} = \gamma$ in either the first half or the second half of the encoded part.

By the definition of a (γ, β, q, L)-ALLED code, with probability at least $2/3$, one of the algorithms $T_1, T_2, \ldots, T_L$ returned by the approximate local list decoder provides oracle access to $\pi(y)$ with at most β fraction of errors. Let T_k be that algorithm. The tester discards this k only if an erroneous point is queried in some iteration of Steps 10-13. Since each proof query of V (in Step 12) is made to a specific index in the proof with probability at most $2/|p(n)|$ and the string decoded by T_k is β-erroneous, by the union bound over queries of V, the probability of V querying an erroneous point is at most $2\beta \cdot \frac{c_2 \cdot 75}{24\varepsilon}$. Hence, by the union bound, the probability that the tester discards k is at most $\frac{1}{3} + 2 \cdot 6 \cdot \lceil \ln 6L \rceil \cdot \frac{c_2 \cdot 75}{24\varepsilon} \cdot \beta \leq \frac{2}{5}$, where the inequality follows from our setting of β. Hence, Step 14 does not reject with probability at least $3/5$. That is, the tester accepts x with probability at least $3/5$.

Assume now that x is ε-far from $\mathcal{P}''$. Let $\mathcal{N}_{\text{pl}}$ denote the set of nonerased points in the plain part of x. Let $\mathcal{N}_{\text{en}}$ denote the set of nonerased points in the encoded part of x. Let α_{pl} denote the fraction (with respect to $s \cdot n$, the length of the plain part) of erased points in the plain part.

Let E denote the event that the number of queries made by the tester does not exceed the query budget Q. In what follows, we upper bound the probability that Algorithm 6.3 accepts, conditioned on E. We prove later, in Claim 6.3.17, that $\Pr[\overline{E}] \leq 1/30$.

Let ε_{pl} denote the fraction of points (with respect to $s \cdot n$, the length of the plain part) in the plain part whose values need to be changed in order to make the plain part a repetition of some string $y \in \{0,1\}^n$. Let $S_a = \{(i-1)n + a : i \in [s]\}$ for all $a \in [n]$. We use the term a-**th segment** to refer to the set S_a. For all $a \in [n]$, we have $|S_a| = s$. For all $a \in [n]$, let $\alpha_a = |\{u \in S_a : x[u] = \perp\}|/s$ denote the fraction of points in S_a that are erased. Let $\mathcal{N}_a \subseteq S_a$ denote the set of nonerased points in the a-th segment.

Case I: the plain part of x is, for every $y \in \{0,1\}^n$, $\varepsilon/144$-far from s repetitions of y.

Let ε_a for all $a \in [n]$ denote the smallest fraction of points in S_a whose values need to be changed in order to satisfy $x[u] = x[v]$ for all $u, v \in \mathcal{N}_a$. For every $a \in [n]$ and $u \in \mathcal{N}_a$, the number of $v \in \mathcal{N}_a$ such that $x[u] \neq x[v]$, is at least $\varepsilon_a \cdot s$. It is immediate that $\varepsilon_{\text{pl}} \cdot s \cdot n = \sum_{a\in[n]} \varepsilon_a \cdot s$.

Let F denote the event that the tester rejects in a single iteration of the loop in Steps 2-7. Let G_a for all $a \in [n]$ denote the event that the tester samples a nonerased point u from S_a in Step 3. Conditioned on G_a, the number of nonerased points in S_a that make the tester reject is at least $\varepsilon_a \cdot s$. Putting all this together, we have,

$$\Pr[F|E] = \sum_{a\in[n]} \Pr[G_a|E] \cdot \Pr[F|G_a, E] = \sum_{a\in[n]} \frac{|\mathcal{N}_a|}{sn} \cdot \frac{\varepsilon_a \cdot s}{|\mathcal{N}_a|} = \sum_{a\in[n]} \frac{1}{n} \cdot \varepsilon_a = \varepsilon_{\text{pl}} \geq \frac{\varepsilon}{144}.$$

Therefore, conditioned on E, in at least $432/\varepsilon$ iterations, the tester will reject with probability at least 19/20. Hence, in Case I, the algorithm accepts with probability at most $\frac{1}{20} + \Pr[\overline{E}] \leq \frac{1}{20} + \frac{1}{30} \leq \frac{2}{5}$, where we prove later (in Claim 6.3.17) $\Pr[\overline{E}] \leq 1/30$. Thus, the algorithm rejects with probability at least 3/5.

Case II: the plain part of x is $\varepsilon/144$-close to s repetitions of a string $y^* \in \{0,1\}^n$. We first show that y^* has to be far from $\mathcal{R}_{\phi_n}$.

Claim 6.3.15. *The string y^* is $\varepsilon/2$-far from $\mathcal{R}_{\phi_n}$.*

Proof. Otherwise, one can transform the entire plain part of x to (be completable to) repetitions of y^* by making at most $sn \cdot \frac{\varepsilon}{144} \leq N \cdot \frac{\varepsilon}{144}$ changes. This can then be transformed to repetitions of a string in $\mathcal{R}_{\phi_n}$ by making at most $sn \cdot \frac{\varepsilon}{2} \leq N \cdot \frac{\varepsilon}{2}$ changes. Thus, the string x can be made to satisfy $\mathcal{P}''$ by making at most $N \cdot \left(\frac{\varepsilon}{144} + \frac{\varepsilon}{2} + \frac{1}{m+1}\right)$ changes, where the term $\frac{N}{m+1}$ accounts for the number of changes in the encoded part. Since $\varepsilon > \frac{\varepsilon^\star}{8}$ and $c_2 > 1$, we have that $m = \frac{28800 \cdot c_2}{\varepsilon^\star} > \frac{144}{71\varepsilon}$. Hence, $\frac{1}{m} < \frac{71\varepsilon}{144}$ and, therefore, $N \cdot \left(\frac{\varepsilon}{144} + \frac{\varepsilon}{2} + \frac{1}{m+1}\right) < \varepsilon N$. Thus, the string x can be made to satisfy $\mathcal{P}''$ by making less than εN changes. This is a contradiction. □

Let $B_i = \{(i-1)n + a : a \in [n]\}$ for all $i \in [s]$. We use the term i-th block to refer to the set B_i. For all $i \in [s]$, we have, $|B_i| = n$. Let $\alpha_i = \frac{|\{u \in B_i : x[u] = \perp\}|}{n}$ for all $i \in [s]$ denote the fraction of points in B_i that are erased. Let $\mathcal{N}_i \subseteq S_i$ denote the set of nonerased points in the i-th block. Let ε_i for all $i \in [s]$ denote the fraction of points in S_i whose values need to be changed in order to satisfy $x[(i-1)n + a] = y^*[a]$ for all $a \in [n]$. In other words, $\varepsilon_i n$ is the smallest number of points in $\mathcal{N}_i$ that need to be changed in order for the i-th block to be completable to y^*.

Fix $k \in [L]$. We show that Algorithm 6.3 discards k with high probability. Consider a single iteration of the repeat-loop in Steps 11-13. Let y' denote the (partially erased) string represented by the block that Algorithm 6.3 samples in Step 11. Let G_1 denote the (good) event that y' is $\varepsilon/6$-close to y^*. Let G_2 denote the (good) event that y' has at most 48α fraction of erasures. We first evaluate the probability that the tester discards k in Steps 11-13 conditioned on G_1 and G_2.

Claim 6.3.16. *Conditioned on G_1 and G_2, the string y' is $24\varepsilon/75$-far from $\mathcal{R}_{\phi_n}$.*

Proof. Let y'' be a string in $\mathcal{R}_{\phi_n}$ closest to y'. Let d denote the number of nonerased bits in y' that need to be changed in order for it to be completable to y''. By our conditioning, y' is a 48α-erased string that is $\varepsilon/6$-close to y^*. Thus, one can convert y^* into y' and then y' into y'' by modifying at most $48\alpha n + \frac{\varepsilon n}{6} + d$ bits in y^*. Since y^* is $\varepsilon/2$-far from $\mathcal{R}_{\phi_n}$, we get that $d \geq \frac{\varepsilon n}{2} - \frac{\varepsilon n}{6} - 48\alpha n$. From the restrictions on α and ε, one can verify that for all settings of these parameters, we have $\alpha \leq \frac{\varepsilon}{3600}$, which implies that $d \geq \frac{24\varepsilon n}{75}$. □

The smooth PCPP V, with proximity parameter $\frac{24\varepsilon}{75}$, is run on y' and the proof decoded by T_k. Let B_1 denote the (bad) event that the PCPP V obtains an erased bit as the answer to some query. Let B_2 denote the (bad) event that V accepts. By Lemma 6.3.9, V makes $\frac{c_2 \cdot 75}{24\varepsilon}$ queries and each query of V to the input part is made to each of the n input indices with probability $\frac{1}{n}$. Hence $\Pr[B_1|E, G_1, G_2]$, the probability that some input query is made to an erased point, is at most $\frac{c_2 \cdot 75}{24\varepsilon} \cdot 48\alpha$.

The probability that the V accepts (even if there were no erased query answers) is $\Pr[B_2|E, G_1, G_2]$ and is, by Definition 6.2.3, at most $1/3$. Thus, the probability that the smooth PCPP accepts, conditioned on E, G_1, and G_2, is by the union bound, at most $\frac{c_2 \cdot 75}{24\varepsilon} \cdot 96\alpha + \frac{1}{3} \leq \frac{1}{24} + \frac{1}{3}$, where the inequality follows from our setting of ε and α.

To bound the probability that the PCPP accepts in a single iteration of Steps 11-13, we now evaluate $\Pr[\overline{G_1}]$ and $\Pr[\overline{G_2}]$. Let the random variable X denote the relative Hamming distance of y' from y^*. Then, $\mathbb{E}[X] = \sum_{i \in [s]} \frac{1}{s} \cdot \varepsilon_i = \varepsilon_{\text{pl}} \leq \frac{\varepsilon}{144}$. By Markov's inequality, $\Pr[\overline{G_1}] = \Pr[X \geq \frac{\varepsilon}{6}] \leq \mathbb{E}[X]/(\varepsilon/6) \leq 1/24$.

To bound $\Pr[\overline{G_2}]$, let the random variable Y denote the fraction of erasures in y'. We have that $\mathbb{E}[Y] = \sum_{i \in [s]} \frac{\alpha_i}{s} = \alpha_{\text{pl}}$. Even if all the erasures were in the plain part, $\alpha_{\text{pl}} \leq \frac{\alpha N}{sn} \leq \alpha \cdot (1 + \frac{1}{m})$. Again, by an application of Markov's inequality, we get $\Pr[\overline{G_2}] = \Pr[Y > 48\alpha] \leq \frac{\mathbb{E}[Y]}{48\alpha} \leq \frac{1+\frac{1}{m}}{48} \leq 1/24$.

Therefore, conditioned on E, the probability that the PCPP accepts in one iteration of Steps 11-13 is at most $\Pr[B_1|E, G_1, G_2] + \Pr[B_2|E, G_1, G_2] + \Pr[\overline{G_2}] + \Pr[\overline{G_1}] \leq \frac{1}{24} + \frac{1}{3} + \frac{1}{24} + \frac{1}{24} \leq \frac{2}{3}$. That is, conditioned on E, for a fixed $k \in [L]$, in $\lceil 6 \ln 6L \rceil$ independent repetitions of Steps 11-13, the probability that the PCPP does not discard k is at most $\left(1 - \frac{1}{3}\right)^{\lceil 6 \ln 6L \rceil} \leq \frac{1}{36L^2}$. Hence, conditioned on E, the probability that for some $k \in [L]$, Steps 10-13 accepts is, by the union bound, at most $1/36L$. Thus, if x is in Case II, the probability that the tester accepts is at most, $\frac{1}{36L} + \Pr[\overline{E}] \leq \frac{1}{36L} + \frac{1}{30} \leq \frac{2}{5}$, where Claim 6.3.17 shows that $\Pr[\overline{E}]$ is at most $1/30$, which then completes the proof of Lemma 6.3.14.

Claim 6.3.17. *The probability that Algorithm 6.3 exceeds its query budget is at most* $1/30$*.*

Proof. We first compute the expected number of queries that the tester makes. The number of queries made in Steps 9-13 is at most $L\lceil 6\ln 6L\rceil \cdot \frac{c_2 \cdot 75}{24\varepsilon} \cdot q$, where q and L are the query complexity and list size of the approximate local list decoder, respectively.

We now calculate the expected number of queries made from Steps 3-7. Let V denote the number of queries made in a particular iteration of Steps 3-7. The variable V is nonzero only if the sampled point u is nonerased. To calculate $\mathbb{E}[V]$:

$$\mathbb{E}[V] = \sum_{a\in[n]} \frac{|\mathcal{N}_a|}{sn} \cdot \frac{1}{1-\alpha_a} = \sum_{a\in[n]} \frac{(1-\alpha_a)s}{sn} \cdot \frac{1}{1-\alpha_a} = 1.$$

Hence, the expected number of queries made by the tester in Steps 2-7 is $\lceil \frac{432}{\varepsilon} \rceil$. Hence, setting Q to $30 \cdot \left(\lceil \frac{432}{\varepsilon} \rceil + L\lceil 6\ln 6L\rceil \cdot \frac{c_2\cdot 75}{24\varepsilon} \cdot q\right)$, and applying Markov's inequality, one can see that $\Pr[\overline{E}] \leq 1/30$. □

□

Next, we show that it is hard to tolerant test $\mathcal{P}''$. The proof of Lemma 6.3.18 is identical to the proof of Lemma 6.3.6 up to change in parameters.

Lemma 6.3.18. *Let* $\varepsilon^\star \in (0,1)$ *be as in Lemma 6.2.2 and* $c_2 > 1$ *be as in Lemma 6.3.9. For every* $\alpha \in (\frac{\varepsilon^\star}{57600\cdot c_2 + 2\varepsilon^\star}, 1)$*, and* $\varepsilon' \in \left(\alpha, \frac{28800\cdot c_2\cdot\varepsilon^\star}{28800\cdot c_2 + \varepsilon^\star}\right)$*, every* (α,ε')*-tolerant tester for* $\mathcal{P}''$ *requires* $\tilde{\Omega}(N^{0.2})$ *queries.*

Proof. Let $\aleph$ be as in Lemma 6.2.2 and let $n \in \aleph$. We will prove the lemma by showing a reduction from $\varepsilon^\star$-testing of $\mathcal{R}_{\phi_n}$. Let N and $p(n)$ be as in Definition 6.3.13. Let s denote $m \cdot 2f(\gamma,\beta)\cdot (p(n))^5/n$.

Consider a string $y \in \{0,1\}^n$ that we want to $\varepsilon^\star$-test for $\mathcal{R}_{\phi_n}$. Let $x \in \{0,1\}^N$ be the string where the first sn bits of x are s repetitions of y and the remaining bits are all 0s. We refer to the substring $x[1 \dots sn]$ as the plain part of x and the substring $x[sn+1 \dots N]$ as the encoded part of x.

Assume that A is an (α, ε')-tolerant tester for $\mathcal{P}''$. We now describe an $\varepsilon^\star$-tester A' for $\mathcal{R}_{\phi_n}$ that has the same query complexity as A. Given oracle access to $y \in \{0,1\}^n$, the tester A' runs the tester A on the string $x \in \{0,1\}^N$ (as constructed from y above) and accepts iff A accepts. Observe that one can simulate a query to x by making at most one query to y.

If $y \in \mathcal{R}_{\phi_n}$, then x is α-close to $\mathcal{P}''$. Observe that the encoded part of x needs to be changed in at most $\frac{1}{2}$ fraction of its positions in order to make it the encoding of a string π, where π is a proof that makes a smooth PCPP for testing $\mathcal{R}_{\phi_n}$ (as guaranteed by Lemma 6.3.9) accept. This follows from the fact that the encoded part of every string that satisfies the property contains an equal number of 0s and 1s. Thus, the fraction of bits in x that needs to be changed in order to make it satisfy $\mathcal{P}''$ is at most $\frac{1}{2} \cdot \frac{N-sn}{N} = \frac{1}{2(m+1)} = \frac{\varepsilon^\star}{57600 \cdot c_2 + 2\varepsilon^\star}$, which is less than α. Therefore, by definition, A' will accept x with probability at least $\frac{2}{3}$.

Assume now that y is $\varepsilon^\star$-far from $\mathcal{R}_{\phi_n}$. Then x needs to be changed in at least $\varepsilon^\star \cdot sn$ positions to make it satisfy $\mathcal{P}''$. From this, one can observe that x is $\varepsilon^\star \cdot \frac{m}{m+1}$-far from $\mathcal{P}''$. Hence, for all $\varepsilon' < \varepsilon^\star \cdot \frac{m}{m+1}$, we have that A will reject x with probability at least $2/3$, and therefore A' will reject y with probability at least $2/3$.

Thus, we have shown that the query complexity of (α, ε')-tolerant testing $\mathcal{P}''$ is at least the query complexity of $\varepsilon^\star$-testing $\mathcal{R}_{\phi_n}$. Hence, the query complexity of (α, ε')-tolerant testing $\mathcal{P}''$ is $\Omega(n)$, which is equal to $\tilde{\Omega}(N^{0.2})$. $\square$

Proof of Theorem 6.1.3. From Theorem 6.3.7, we get the following constraints on ε and α:

$$\frac{\varepsilon^\star}{57600 \cdot c_2 + 2\varepsilon^\star} < \alpha < \frac{\varepsilon^\star}{57600 \cdot c_2}; \qquad \varepsilon > \frac{\varepsilon^\star}{8}; \qquad \varepsilon' = \alpha + \varepsilon < \frac{28800 \cdot c_2 \varepsilon^\star}{28800 \cdot c_2 + \varepsilon^\star}.$$

For sufficiently small $\varepsilon^\star$, the upper bound on $\varepsilon+\alpha$ is strictly greater than $\varepsilon^\star/2$. So, it is enough to find $\varepsilon < \varepsilon^\star/4$ and $\alpha < \varepsilon^\star/4$ that also satisfy the first two conditions. The existence of such ε and α is clear from the bounds imposed on them by the first two constraints. $\square$

Algorithm 6.3 Erasure-resilient tester for separating property $\mathcal{P}''$

```
Input: $\alpha, \varepsilon \in (0,1), N = (m+1) \cdot 2f(\gamma,\beta) \cdot (p(n))^5$; oracle access to $x \in \{0,1,\perp\}^N$
    ▷ Set $s \leftarrow m \cdot \frac{2f(\gamma,\beta)\cdot(p(n))^5}{n}$, $q \leftarrow \frac{c_3 \log(1/\beta)}{(1-\gamma)^3}$, and $L \leftarrow \frac{c_3}{(1-\gamma)^2}$.
    ▷ Set the query budget $Q \leftarrow 30 \cdot \left(\lceil\frac{432}{\varepsilon}\rceil + L\lceil 6\ln 6L\rceil \cdot \frac{c_2\cdot 75}{24\varepsilon} \cdot q\right)$.
 1: Accept whenever the number of queries exceeds $Q$.
    ▷ Steps 2-7 check that the plain part of $x$ is the repetition of a string $y \in \{0,1\}^n$.
 2: repeat $\lceil\frac{432}{\varepsilon}\rceil$ times:
 3:     Sample a uniformly random point $u$ from the plain part.
 4:     if $x[u] \neq \perp$ then
 5:         Let $i \in [s]$, $a \in [n]$ be such that $u = (i-1)\cdot n + a$.
 6:         Repeatedly sample $j \in [s]$ uniformly at random until $x[(j-1)n+a] \neq \perp$.
 7:         Reject if $x[u] \neq x[(j-1)n+a]$.
    ▷ In order to query the $i$-th bit of the encoding, we query the $i$-th bits of both the first and second halves of the encoded part. We set the $i$-th bit of the encoding to the $i$-th bit of the first half if that is nonerased, and to the complement of the $i$-th bit of second half if that is nonerased. If both are erased, we set the $i$-th bit of the encoding to $\perp$.
 8: Run the decoder for the $(\gamma,\beta,q,L)$-ALLED code (from Lemma 6.3.12) with oracle access to the encoded part of $x$.
    ▷ Let $A_1, A_2, \ldots, A_L$ be the list of algorithms returned in the above step.
    ▷ Steps 9-14 check that $y \in \mathcal{R}_{\phi_n}$ using the smooth PCPP $V$ (from Lemma 6.3.9) on decoded proofs.
 9: for each $k \in [L]$ do
10:     repeat $\lceil 6\ln 6L\rceil$ times:
11:         Sample $i \in [s]$ uniformly at random.
12:         Run the smooth PCPP $V$ with proximity parameter $\frac{24\varepsilon}{75}$, and oracle access to the concatenation of $x[(i-1)\cdot n+1,\ldots,(i-1)\cdot n+n]$ and the string decoded by $T_k$.
13:         Discard the current $k$ if all query answers to $V$ are nonerased and $V$ rejects.
14: Reject if every $k \in [L]$ is discarded; otherwise, accept.
```

CHAPTER 7
Average Sensitivity of Graph Algorithms

In this chapter, we define the notion of average sensitivity and provide algorithms with low average sensitivity for several graph problems.

7.1 AVERAGE SENSITIVITY: OUR DEFINITION

We assume that the n-node input graph G' at hand is a randomly chosen (large) subgraph of an unknown true graph G. Intuitively, a deterministic algorithm $\mathcal{A}$ is stable-on-average when the Hamming distance $d_{\mathrm{Ham}}\big(\mathcal{A}(G), \mathcal{A}(G')\big)$ is small, where $\mathcal{A}(G)$ and $\mathcal{A}(G')$ are outputs of $\mathcal{A}$ on G and G', respectively. Here, outputs are typically vertex sets or edges sets and we assume that they are represented appropriately using binary strings.

More specifically, for an integer $k \geq 1$, we say that the *k-average sensitivity* of a deterministic algorithm $\mathcal{A}$ is

$$\mathbb{E}_{\{e_1,\ldots,e_k\}\sim\binom{E}{k}}\left[d_{\mathrm{Ham}}\big(\mathcal{A}(G), \mathcal{A}(G - \{e_1, \ldots, e_k\})\big)\right] \tag{7.1}$$

for every graph $G = (V, E)$, where $G - F$ for an edge set F is the subgraph obtained from G by removing F, and $e_1, \ldots, e_k$ are sampled from E uniformly at random without replacement. When $k = 1$, we call the k-average sensitivity simply the *average sensitivity*. We say that algorithms with low k-average sensitivity are *k-stable-on-average*. Although we focus on graphs here, our definition can also be extended to the study of combinatorial objects other than graphs such as strings and constraint satisfaction problems.

An algorithm that outputs the same solution regardless of the input has the least possible average sensitivity, though it is definitely useless. The key question in our study of average sensitivity, therefore, is to understand the trade-off between the solution quality and the average sensitivity for various problems.

Example 7.1.1. *Consider the algorithm that, given a graph $G = (V, E)$ on n vertices, outputs the set of vertices of degree at least $n/2$. As removing an edge changes the degree of exactly two vertices, the sensitivity of this algorithm is at most 2.*

Example 7.1.2. *Consider the s-t shortest path problem, where given a graph G=(V,E) and two vertices $s, t \in V$, we are to output the set of edges in a shortest path from s to t. Since the length of a shortest path is always bounded by n, where n is the number of vertices, every deterministic algorithm has average sensitivity $O(n)$. Indeed, there exists a graph for which this trivial upper bound is tight. Think of a cycle of even length n and two vertices s, t in diametrically opposite positions. Consider an arbitrary deterministic algorithm $\mathcal{A}$, and assume that it outputs a path P (of length $n/2$) among the two shortest paths from s to t. With probability half, an edge in P is removed, and $\mathcal{A}$ must output the other path Q (of length $n/2$) from s to t. Hence, the average sensitivity must be $1/2 \cdot (n/2) = \Omega(n)$. In this sense, there is no deterministic algorithm with nontrivial average sensitivity for the s-t shortest path problem.*

We also generalize our definition of average sensitivity to apply to randomized algorithms. Let $\mathcal{A}(G)$ denote the output distribution of $\mathcal{A}$ on G. Let $d_{\mathrm{EM}}(\mathcal{A}(G), \mathcal{A}(G'))$ denote the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G')$, where the distance between two outputs is measured by the Hamming distance. Then, for an integer $k \geq 1$, we say that the *k-average sensitivity* of a randomized algorithm $\mathcal{A}$ is

$$\mathbb{E}_{\{e_1,\ldots,e_k\} \sim \binom{E}{k}} \left[d_{\mathrm{EM}}\big(\mathcal{A}(G), \mathcal{A}(G - \{e_1, \ldots, e_k\})\big) \right], \tag{7.2}$$

where $e_1, \ldots, e_k$ are sampled from E uniformly at random without replacement. Note that when the algorithm $\mathcal{A}$ is deterministic, (7.2) matches the definition of the average sensitivity for deterministic algorithms.

Remark 7.1.3. *The k-average sensitivity of an algorithm $\mathcal{A}$ with respect to the total variation distance can be defined as $\mathbb{E}_{\{e_1,\ldots,e_k\}\sim\binom{E}{k}}\left[d_{\mathrm{TV}}\big(\mathcal{A}(G), \mathcal{A}(G-\{e_1,\ldots,e_k\})\big)\right]$, where $d_{\mathrm{TV}}(\cdot,\cdot)$ denotes the total variation distance. It is easy to observe that, if the k-average sensitivity of an algorithm with respect to the total variation distance is at most $\gamma(G)$, then its k-average sensitivity is bounded by $\mathsf{H}\cdot\gamma(G)$, where the H is the maximum Hamming weight of a solution output by $\mathcal{A}$ on G.*

Example 7.1.4. *Randomness does not help improve the average sensitivity of algorithms for the s-t shortest path problem. Think of the cycle graph given in Example 7.1.2, and suppose that a randomized algorithm $\mathcal{A}$ outputs P and Q with probability p and $q = 1-p$, respectively. Then, the average sensitivity is $p\cdot 1/2\cdot(n/2)+q\cdot 1/2\cdot(n/2)=\Omega(n)$.*

7.2 PROPERTIES OF AVERAGE SENSITIVITY AND OUR RESULTS

In this section, we discuss various nice properties of our definition of average sensitivity and state our bounds on average sensitivity of algorithms for several graph problems.

Basic Properties of Average Sensitivity. The definition of average sensitivity has many nice properties. Here we discuss some useful properties of average sensitivity that we use as building blocks in the design of our stable-on-average algorithms. We denote by $\mathcal{G}$ the (infinite) set consisting of all graphs. Given a graph $G=(V,E)$ and an edge $e\in E$, we use G^{-e} as a shorthand for $G-\{e\}$.

Bounds on k-average sensitivity from bounds on average sensitivity. This is one of the most important properties of our definition of average sensitivity. It essentially says that bounding the average sensitivity of an algorithm with respect to the removal of a single edge automatically gives a bound on the average sensitivity of that algorithm with respect to the removal of multiple edges. In other words, it is enough to analyze the average sensitivity of an algorithm with respect to the removal of a single edge.

Theorem 7.2.1. *Let $\mathcal{A}$ be an algorithm for a graph problem with the average sensitivity given by $f(n, m)$. Then, for any integer $k \geq 1$, the algorithm $\mathcal{A}$ has k-average sensitivity at most $\sum_{i=1}^{k} f(n, m-i+1)$.*

In particular, if the average sensitivity is a nondecreasing function of the number of edges, the above theorem immediately implies that the k-average sensitivity is at most k times the average sensitivity.

Sequential composition. It will be useful if we can obtain a stable-on-average algorithm by sequentially applying several stable-on-average subroutines. We show two different sequential composition theorems for average sensitivity.

Theorem 7.2.2 (Sequential composition)**.** *Consider two randomized algorithms $\mathcal{A}_1 : \mathcal{G} \to \mathcal{S}_1, \mathcal{A}_2 : \mathcal{G} \times \mathcal{S}_1 \to \mathcal{S}_2$. Suppose that the average sensitivity of $\mathcal{A}_1$ with respect to the total variation distance is γ_1 and the average sensitivity of $\mathcal{A}_2(\cdot, S_1)$ is $\beta_2^{(S_1)}$ for any $S_1 \in \mathcal{S}_1$. Let $\mathcal{A} : \mathcal{G} \to \mathcal{S}_2$ be a randomized algorithm obtained by composing $\mathcal{A}_1$ and $\mathcal{A}_2$, that is, $\mathcal{A}(G) = \mathcal{A}_2(G, \mathcal{A}_1(G))$. Then, the average sensitivity of $\mathcal{A}$ is $\mathsf{H} \cdot \gamma_1(G) + \mathbb{E}_{S_1 \sim \mathcal{A}_1(G)}\left[\beta_2^{(S_1)}(G)\right]$, where H denotes the maximum Hamming weight among those of solutions obtained by running $\mathcal{A}$ on G and G^{-e} over all $e \in E$.*

Our second composition theorem is for the average sensitivity with respect to the total variation distance. This is also useful for analyzing the average sensitivity with respect to the earth mover's distance, as it can be bounded by the average sensitivity with respect to the total variation distance times the maximum Hamming weight of a solution, as in Remark 7.1.3.

Theorem 7.2.3 (Sequential composition w.r.t. the TV distance)**.** *Consider k randomized algorithms $\mathcal{A}_i : \mathcal{G} \times \prod_{j=1}^{i-1} \mathcal{S}_j \to \mathcal{S}_i$ for $i \in [k]$. Suppose that, for each $i \in [k]$, the average sensitivity of $\mathcal{A}_i(\cdot, S_1, \dots, S_{i-1})$ is γ_i with respect to the total variation distance for every $S_1 \in \mathcal{S}_1, \dots, S_{i-1} \in \mathcal{S}_{i-1}$. Consider a sequence of computations $S_1 = \mathcal{A}_1(G), S_2 = \mathcal{A}_2(G, S_1), \dots, S_k = \mathcal{A}_k(G, S_1, \dots, S_{k-1})$. Let $\mathcal{A} : \mathcal{G} \to \mathcal{S}_k$ be a randomized algorithm that performs this sequence of computations on input G and outputs S_k. Then, the average sensitivity of $\mathcal{A}$ with respect to the total variation distance is at most $\sum_{i=1}^{k} \gamma_i(G)$.*

Parallel composition. It is often the case that there are multiple algorithms that solve the same problem albeit with different average sensitivity guarantees. Such stable-on-average algorithms can be composed by running them according to a distribution determined by the input graph. The advantage of such a composition, which we call a parallel composition, is that the average sensitivity of the resulting algorithm might be better than that of the component algorithms.

Theorem 7.2.4 (Parallel composition)**.** *Let $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_k$ be algorithms for a graph problem with average sensitivity $\beta_1, \beta_2, \dots, \beta_k$, respectively. Let $\mathcal{A}$ be an algorithm that, given a graph G, runs $\mathcal{A}_i$ with probability $\rho_i(G)$ for $i \in [k]$, where $\sum_{i \in [k]} \rho_i(G) = 1$. Let H denote the maximum Hamming weight among the solutions obtained by running $\mathcal{A}$ on G and $\{G^{-e}\}_{e \in E}$. Then the average sensitivity of $\mathcal{A}$ is at most $\sum_{i \in [k]} \rho_i(G) \cdot \beta_i(G) + \mathsf{H} \cdot \mathbb{E}_{e \sim E}\left[\sum_{i \in [k]} |\rho_i(G) - \rho_i(G^{-e})|\right]$.*

We use Theorem 7.2.4 extensively to combine algorithms with different average sensitivity.

Connection to Local Algorithms. We show a relationship between the average sensitivity of an algorithm and the query complexity of a local algorithm that simulates oracle access to the solution output by the former algorithm. Roughly speaking, we show, in Theorem 7.2.5, that the average sensitivity of an algorithm $\mathcal{A}$ is bounded by the query complexity of another algorithm $\mathcal{O}$, where $\mathcal{O}$ makes queries to a graph G and simulates oracle access to the solution produced by $\mathcal{A}$ on input G. We use Theorem 7.2.5 to prove the existence of stable-on-average matching algorithms based on the sublinear-time matching algorithms due to Yoshida et al. (2012).

Theorem 7.2.5 (Locality implies low average sensitivity)**.** *Consider a randomized algorithm $\mathcal{A} : \mathcal{G} \to \mathcal{S}$ for a graph problem, where each solution output by $\mathcal{A}$ is a subset of the set of edges of the input graph. Assume that there exists an oracle $\mathcal{O}$ satisfying the following:*

- *when given access to a graph $G = (V, E)$ and query $e \in E$, the oracle generates a random string $\pi \in \{0,1\}^{r(|V|)}$ and outputs whether e is contained in the solution obtained by running $\mathcal{A}$ on G with π as its random string,*
- *the oracle $\mathcal{O}$ makes at most $q(G)$ queries to G in expectation, where this expectation is taken over the random coins of $\mathcal{A}$ and a uniformly random query $e \in E$.*

Then, $\mathcal{A}$ has average sensitivity at most $q(G)$.

Moreover, given the promise that the input graphs satisfy $|E| \geq |V|$, the statement applies also to algorithms for which each solution is a subset of the vertex set of the input graph.

Theorem 7.2.5 cements the intuition that strong locality guarantees for solutions output by an algorithm imply that the removal of edges from a graph affects only the presence of a few edges in the solution, which in turn implies low average sensitivity. Due to its applicability in bounding the average sensitivity of algorithms, we think that Theorem 7.2.5 could lead to further research in the design of local algorithms for various graph problems.

Stable-on-average Algorithms for Concrete Problems. We summarize, in Table 7.1, the average sensitivity bounds that we obtain for various concrete problems. All our algorithms run in polynomial time, and the bounds on k-average sensitivity of these algorithms can be easily derived using Theorem 7.2.1. To help interpret our bounds on average sensitivity, we mention that for maximization problems whose optimal values are sufficiently Lipschitz with respect to edge removals, $O(\mathsf{OPT})$ is a trivial upper bound for the average sensitivity. However, this is not the case, in general, for minimization problems.

For the minimum spanning forest problem, we show that Kruskal's algorithm (Kruskal (1956)) achieves average sensitivity $O(n/m)$, which is quite small regarding that the spanning forest can have $\Omega(n)$ edges. In contrast, it is not hard to show that the average sensitivity of many of the known polynomial-time (approximation) algorithms for the other problems listed in Table 7.1 are all $\Omega(n)$.

For the global minimum cut problem, our algorithm outputs a cut as a vertex set. As the approximation ratio of our algorithm is constant, it is likely to output a cut of size close to OPT, and hence we want to make its average sensitivity smaller than OPT. We observe that the average sensitivity becomes smaller than OPT when $\mathsf{OPT} = \Omega(t \log\log t/ \log t)$ for $t = \log(n)/\varepsilon$, and it quickly decreases as OPT increases.

For the maximum matching problem, the average sensitivity is $\Omega(n)$ if we want to

Table 7.1: Our results on average sensitivity. Here n, m, OPT denote the number of vertices, the number of edges, and the optimal value, respectively, and $\varepsilon \in (0,1)$ is an arbitrary constant. The notation $\widetilde{O}(\cdot)$ hides a polylogarithmic factor in n.

Problem	Approximation Ratio	Average Sensitivity	Reference
Min Spanning Forest	1	$O\left(\frac{n}{m}\right)$	Section 7.6
Global Minimum Cut	$2+\varepsilon$	$n^{O\left(\frac{1}{(\varepsilon\cdot\mathsf{OPT})}\right)}$	Section 7.7.1
Max Matching	$1/2 - o(1)$	$\widetilde{O}(\mathsf{OPT}^{3/4})$	Section 7.8.2
	$1-\varepsilon$	$\widetilde{O}\left(\left(\frac{\mathsf{OPT}}{\varepsilon^3}\right)^{\frac{1}{(1+\Omega(\varepsilon^2))}}\right)$	Section 7.8.3
	1	$\Omega(n)$	Section 7.8.4
Min Vertex Cover	$2+o(1)$	$\widetilde{O}(\mathsf{OPT}^{3/4})$	Section 7.9.1
	$\widetilde{O}\left(\frac{m^{1+\varepsilon}}{n}\right)$	$O\left(\frac{n^2}{m^{1+\varepsilon}}\right)$	Section 7.9.2
2-Coloring	—	$\Omega(n)$	Section 7.10

output the exact maximum matching. We propose two stable-on-average approximation algorithms for maximum matching. Our first algorithm has approximation ratio $1/2 - o(1)$ and average sensitivity $\widetilde{O}(\mathsf{OPT}^{3/4})$. The second one has approximation ratio $1-\varepsilon$ and average sensitivity $\widetilde{O}\left((\mathsf{OPT}/\varepsilon^3)^{1/(1+\Omega(\varepsilon^2))}\right)$ for every constant $\varepsilon \in (0,1)$, which shows that we do not have to sacrifice the approximation ratio a lot to obtain a nontrivial average sensitivity.

For the minimum vertex cover problem, we propose two algorithms. The first algorithm has approximation ratio $2+o(1)$, which is close to the best we can hope for as obtaining $(2-\varepsilon)$-approximation is NP-Hard assuming the Unique Games conjecture (Khot & Regev (2003)). Moreover, the average sensitivity of $\widetilde{O}(\mathsf{OPT}^{3/4})$ is much smaller than the trivial $O(\mathsf{OPT})$. The second algorithm has a worse approximation ratio but can achieve a better average sensitivity in some regimes. For example, when $\mathsf{OPT} = \Omega(n)$, $m = \Theta(n)$ and $\varepsilon = 1/2$, the average sensitivity of the first algorithm is

$\Omega(n^{3/4})$ whereas that of the second algorithm is $O(n^{1/2})$.

In the 2-coloring problem, given a bipartite graph, we are to output one part in the bipartition. For this problem, we show a lower bound of $\Omega(n)$ for the average sensitivity: that is, there is no algorithm with nontrivial average sensitivity.

7.3 DISCUSSIONS ON AVERAGE SENSITIVITY

Output representation. The notion of average sensitivity is dependent on the output representation. For example, we can double the average sensitivity by duplicating the output. A natural idea for alleviating this issue is to normalize the average sensitivity by the maximum Hamming weight H of a solution. However, for minimization problems where the optimal value OPT could be much smaller than H, such a normalization can diminish subtle differences in average sensitivity, e.g., $O(\mathsf{OPT}^{1/2})$ vs $O(\mathsf{OPT})$. It is an interesting open question whether there is a canonical way to normalize average sensitivity so that the resulting quantity is independent of the output representation.

Sensitivity against adversarial edge removals. It is also natural to take the maximum, instead of the average, over edges in definitions (7.1) and (7.2), which can be seen as sensitivity against adversarial edge removals. Indeed a similar notion has been proposed to study algorithms for geometric problems (Meulemans et al. (2018)). However, in the case of graph algorithms, it is hard to guarantee that the output of an algorithm does not change much after removing an arbitrary edge. Moreover, by a standard averaging argument, one can say that for 99% of arbitrary edge removals, the sensitivity of an algorithm is asymptotically equal to the average sensitivity, which is sufficient in most cases.

Average sensitivity w.r.t. edge additions. As another variant of average sensitivity, it is natural to consider incorporating edge additions in definitions (7.1) and (7.2). In contrast to removing edges, it is not always clear how we should add edges to the graph in definitions (7.1) and (7.2). A naive idea is sampling k pairs of vertices uniformly at random and adding edges between them. This procedure makes the graph close to a graph sampled from the Erdős-Rényi model (Erdős & Rényi (1959)), which does not represent real networks such as social networks and road networks well. To avoid this subtle issue, here, we focus on removing edges.

Alternative notion of average sensitivity for randomized algorithms. Consider a randomized algorithm $\mathcal{A}$ that, given a graph G on n vertices, generates a random string $\pi \in \{0,1\}^{r(n)}$ for some function $r : \mathbb{N} \to \mathbb{N}$, and then runs a deterministic algorithm $\mathcal{A}_\pi$ on G, where the algorithm $\mathcal{A}_\pi$ has π hardwired into it. Assume that $\mathcal{A}_\pi$ can be applied to any graph. It is also natural to define the average sensitivity of $\mathcal{A}$ as

$$\mathbb{E}_{e\sim E}\left[\mathbb{E}_\pi\left[d_{\mathrm{Ham}}\big(\mathcal{A}_\pi(G), \mathcal{A}_\pi(G^{-e})\big)\right]\right]. \tag{7.3}$$

In other words, we measure the expected distance between the outputs of $\mathcal{A}$ on G and G^{-e} when we feed the same string π to $\mathcal{A}$, over the choice of π and edge e. Note that (7.3) upper bounds (7.2) because, in the definition of the earth mover's distance, we optimally transport probability mass from $\mathcal{A}(G)$ to $\mathcal{A}(G^{-e})$ whereas, in (7.3), how the probability mass is transported is not necessarily optimal.

We can actually bound (7.3) for some of our algorithms. In this chapter, however, we focus on the definition (7.2) because the assumption that A_π can be applied to any graph does not hold in general, and bounding (7.3) is unnecessarily tedious and is not very enlightening.

7.4 RELATED WORK

Average sensitivity of network centralities. *(Network) centrality* is a collective name for indicators that measure importance of vertices or edges in a network. Notable examples are closeness centrality (Bavelas (1950); Beauchamp (1965); Sabidussi (1966)), harmonic centrality (Marchiori & Latora (2000)), betweenness centrality (Freeman (1977)), and PageRank (Page et al. (1999)). To compare these centralities qualitatively, Murai & Yoshida (2019) recently introduced the notion of average-case sensitivity for centralities. Fix a vertex centrality measure c; let $c_G(v)$ denote the centrality of a vertex $v \in V$ in a graph $G = (V, E)$. Then, the *average-case sensitivity* of c on G is defined as

$$S_c(G) = \mathbb{E}_{e \sim E} \mathbb{E}_{v \sim V} \frac{|c_{G^{-e}}(v) - c_G(v)|}{c_G(v)},$$

where e and v are sampled uniformly at random. Murai & Yoshida (2019) showed various upper and lower bounds for centralities.

Since a centrality measure assigns real values to vertices, they studied the relative change of the centrality values upon removal of random edges. As our focus in this chapter is on graph algorithms, our notion (7.2) measures the Hamming distance between solutions when one removes random edges.

Differential privacy. *Differential privacy* (Dwork et al. (2006)) is a notion closely related to average sensitivity. Assuming the existence of a neighbor relation over inputs, the definition of differential privacy requires that the distributions of outputs on neighboring inputs are similar. The variant of differential privacy closest to our definition of average sensitivity is edge differential privacy introduced by Nissim et al. (2007) and further studied by Hay et al. (2009); Gupta et al. (2010); Karwa & Slavkovic (2012); Kasiviswanathan et al. (2013); Karwa et al. (2014); Raskhodnikova & Smith (2016). Here, the neighbors of a graph $G = (V, E)$ are defined to be $\{G^{-e}\}_{e \in E}$. For $\varepsilon > 0$, an algorithm is *ε-differentially private* if for all $e \in E$,

$$\exp(-\varepsilon) \cdot \Pr[\mathcal{A}(G^{-e}) \in \mathcal{S}] \leq \Pr[\mathcal{A}(G) \in \mathcal{S}] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{A}(G^{-e}) \in \mathcal{S}] \tag{7.4}$$

for any set of solutions $\mathcal{S}$.

Differential privacy has stricter requirements than average sensitivity. For example, an algorithm that outputs a vertex cover of the input graph of size smaller than $n-1$ is not differentially private. This is because the output reveals that there is no edge between two vertices that are not part of the vertex cover. It follows that we can only output a vertex cover of size at least $n-1$. To avoid this issue, Gupta et al. (2010) considered outputting an implicit representation of a vertex cover.

Moreover, since differential privacy guarantees that the probabilities of outputting a specific solution on G and G^{-e} are close to each other, the total variation distance between the two distributions $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$ must be small. Since the earth mover's distance between two output distributions can be small even if the total variation distance between them is large, even if an algorithm does not satisfy the conditions of differential privacy, it could still have small average sensitivity. Despite these differences, our algorithms for the global minimum cut problem and the vertex cover problem are inspired by differentially private algorithms for the same problems (Gupta et al. (2010)).

Generalization and stability of learning algorithms. A similar notion of stability has been defined for learning algorithms and its connection to the generalization error has been explored.

Generalization (Shalev-Shwartz & Ben-David (2009)) is a fundamental concept in statistical learning theory. Given samples $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ from an unknown true distribution $\mathcal{D}$ over a dataset, the goal of a learning algorithm $\mathcal{L}$ is to output a parameter θ that minimizes *expected loss* $\mathbb{E}_{\boldsymbol{z} \sim \mathcal{D}}[\ell(\boldsymbol{z}; \theta)]$, where $\ell(\boldsymbol{z}; \theta)$ is the loss incurred by a sample $\boldsymbol{z}$ with respect to a parameter θ. As the true distribution $\mathcal{D}$ is unknown, a frequently used approach in learning is to compute a parameter θ that minimizes the *empirical loss* $\frac{1}{n} \cdot \sum_{i=1}^{n} \ell(\boldsymbol{z}_i; \theta)$, which is an unbiased estimator of the expected loss and is purely a function of the available samples. The *generalization error* of a learner $\mathcal{L}$ is a measure of how close the empirical loss is to the expected loss as a function of the sample size n.

One technique to reduce the generalization error is to add a *regularization* term to the loss function being minimized (Bousquet & Elisseeff (2002)). This also ensures that the learned parameter θ does not change much with respect to minor changes in the samples being used for learning. Therefore, in a sense, learning algorithms that use regularization can be considered as being stable according to our definition of sensitivity.

Bousquet & Elisseeff (2002) defined a notion of stability for learning algorithms in relation to reducing the generalization error. Their stability notion requires that the empirical loss of the learning algorithm does not change much by removing or replacing *any* sample in the input data. In contrast, in our definition of average sensitivity, we consider removing *random* edges from a graph, and measure the change in the output solution rather than that in the objective value.

7.5 OVERVIEW OF OUR TECHNIQUES

Minimum spanning forest. For the minimum spanning forest problem, we show that the classical Kruskal's algorithm has low average sensitivity; specifically, at most 1. Interestingly, Kruskal's algorithm is deterministic and yet has low average sensitivity. In contrast, we show that Prim's algorithm has average sensitivity $\Omega(m)$ for a natural (and deterministic) rule of breaking ties among edges.

Global minimum cut. For the global minimum cut problem, our algorithm is inspired by a differentially private algorithm by Gupta et al. (2010). Our algorithm, given a parameter $\varepsilon > 0$ and a graph G as input, first enumerates a list of cuts whose sizes are at most $(2+\varepsilon)\cdot\mathsf{OPT}$; this enumeration can be done in polynomial time as shown by Karger (Karger (1993)). It then outputs a cut from the list with probability exponentially small in the product of the size of the cut and $O(1/(\varepsilon\cdot\mathsf{OPT}))$. The main argument in analyzing the average sensitivity of the algorithm is that the aforementioned distribution is close (in earth mover's distance) to a related Gibbs distribution on the set of all cuts in the graph. Therefore, the average sensitivity of the algorithm is of the same order as the average sensitivity of sampling a cut from such a Gibbs distribution doing which requires exponential time. We finally show that the average sensitivity of sampling a cut from this Gibbs distribution is at most $n^{O(1/(\varepsilon\cdot\mathsf{OPT}))}$.

Maximum matching. There are several components to the design and analysis of our stable-on-average $(\frac{1}{2}-\varepsilon)$-approximation algorithm for the maximum matching problem. Our starting point is the observation (Theorem 7.2.5) that the ability to locally simulate access to the solution of an algorithm $\mathcal{A}$ implies that $\mathcal{A}$ is stable-on-average. We use this to bound the average sensitivity of a randomized greedy $\frac{1}{2}$-approximation algorithm $\mathcal{A}$ for the maximum matching problem. Specifically, $\mathcal{A}$ constructs a maximal matching by iterating over edges in the input graph $G = (V, E)$ according to a uniformly random ordering and adding an edge to the current matching if the addition does not violate the matching property. Yoshida et al. (2012) constructed a local algorithm that, given a uniformly random edge $e \in E$ as input, makes $O(\Delta)$ queries to G in expectation and answers whether e is in the matching output by $\mathcal{A}$ on G, where the expectation is over the choice of input e and the randomness in $\mathcal{A}$, and Δ is the maximum degree of G. Combined with Theorem 7.2.5, this implies that the average sensitivity of $\mathcal{A}$ is $O(\Delta)$.

Next, we transform $\mathcal{A}$ to also work for graphs of unbounded degree as follows. The idea is to remove vertices of degree at least $\frac{m}{\varepsilon \cdot \mathsf{OPT}}$ from the graph and run $\mathcal{A}$ on the resulting graph. This transformation affects the approximation guarantee only by an additive $\varepsilon \cdot \mathsf{OPT}$ term as the number of such *high degree* vertices is small. However, this thresholding procedure could in itself have high average sensitivity, since the thresholds of G and G^{-e} are different for any $e \in E$.

We circumvent this issue by using a Laplace random variable L as the threshold, where the distribution of L is tightly concentrated around $\frac{m}{\varepsilon \cdot \mathsf{OPT}}$. We use our sequential composition theorem (Theorem 7.2.2) in order to analyze the average sensitivity of the resulting procedure, where we consider the instantiation of the Laplace random threshold as the first algorithm and the remaining steps in the procedure as the second algorithm. The first term in the expression given by Theorem 7.2.2 turns out to be a negligible quantity and is easy to bound. The main task in bounding the second term is to bound, for all $x \in \mathbb{R}$, the average sensitivity of a procedure $\mathcal{A}_x$ that, on input graph G, removes all vertices of degree at least x from G and runs the randomized greedy maximal matching algorithm. The heart of the argument in bounding this average sensitivity is that given a local algorithm $\mathcal{O}$ with query complexity $q(\Delta)$ that simulates oracle access to the solutions output by an algorithm $\mathcal{A}$, we can, for all $x \in \mathbb{R}$, construct a local algorithm $\mathcal{O}_x$ for the algorithm $\mathcal{A}_x$. Moreover, the query complexity of $\mathcal{O}_x$, which also bounds the average sensitivity of $\mathcal{A}_x$ by Theorem 7.2.5, is at most $O(x^2 q(x))$. This implies that the second term in the expression given by Theorem 7.2.2, is $\mathbb{E}_L\left[O(L^2 q(L))\right] = O((\frac{m}{\varepsilon \mathsf{OPT}})^3)$.

An issue with the aforementioned matching algorithm is that its average sensitivity is poor for graphs with small values of OPT. However, we observe that the algorithm that simply outputs the lexicographically smallest maximum matching does not have this issue. Its average sensitivity is $O(\mathsf{OPT}^2/m)$, since the output matching stays the same unless an edge in the matching is removed. We obtain our final stable-on-average $(\frac{1}{2} - \varepsilon)$-approximation algorithm for the maximum matching problem by running these two algorithms according to a probability distribution determined by the input graph. By our parallel composition theorem, the sensitivity of the resultant algorithm is $O\left((\mathsf{OPT}/\varepsilon)^{3/4}\right)$.

The design and analysis of our stable-on-average $(1-\varepsilon)$-approximation algorithm for the maximum matching problem uses similar ideas as above. The only difference is that we replace the randomized greedy maximal matching algorithm above with a $(1-\varepsilon)$-approximation algorithm that repeatedly improves a matching using greedily chosen augmenting paths.

Minimum vertex cover. We describe two stable-on-average algorithms for the minimum vertex cover problem. Our $(2+\varepsilon)$-approximation algorithm is based on a reduction from the stable-on-average $(\frac{1}{2} - \varepsilon)$-approximation algorithm for the maximum matching problem. In particular, it runs the stable-on-average matching algorithm and outputs a union of the set of vertices removed (by thresholding) and the set of endpoints of the matching computed. For the approximation guarantee, we argue that, with high probability, the cardinality of the set of removed vertices is $O(\varepsilon \mathsf{OPT})$. The main task in showing that the algorithm is stable-on-average is to bound the average sensitivity of outputting the set of removed vertices. In case the same value of threshold is used for G and $G - e$, the cardinality of the symmetric difference between the sets of removed vertices is at most 2. Equipped with this observation and the ideas used in bounding the average sensitivity of our matching algorithms, we bound the average sensitivity of returning the set of removed vertices.

Our second algorithm for the minimum vertex cover problem is based on a differentially private vertex cover approximation algorithm by Gupta et al. (2010). Specifically, we output a permutation of the vertices and for each edge, its first endpoint in the permutation is in the vertex cover. If we generate our permutation by repeatedly sampling vertices according to their yet *uncovered* degree, we get a 2-approximation algorithm for vertex cover (Pitt (1985)). If we instead output a uniformly random permutation of vertices, we get an algorithm with good average sensitivity but poor approximation guarantee. Our algorithm finds a middle ground between these approaches, by selecting vertices with probability proportional to their *uncovered* degrees in the beginning and progressively skewing towards the uniform distribution.

2-coloring. To show our $\Omega(n)$ lower bound on average sensitivity for 2-coloring, consider the set of all paths on n vertices and the set of all graphs obtained by removing exactly one edge from these paths (called 2-part-paths). A path has exactly two ways of being 2-colored and a 2-part-path has four ways of being 2-colored. A path and 2-part-path are neighbors if the latter is obtained from the former by removing an edge. A 2-part-path has at most four neighbors. The output distribution of any

2-coloring algorithm $\mathcal{A}$ on a 2-part-path can be close (in earth mover's distance) only to those of at most 2 of its neighboring paths. If $\mathcal{A}$, however, has low average sensitivity, the output distributions of $\mathcal{A}$ have to be close on a large fraction of pairs of neighboring graphs, which gives a contradiction.

Notation

Let $G = (V, E)$ be a graph. Given a graph $G = (V, E)$, for an edge $e \in E$, we denote by G^{-e} the graph obtained by removing e from G. Similarly, for an edge set $F \subseteq E$, we denote by $G - F$ the graph obtained by removing every edge in F from G. For an edge set $F \subseteq E$, let $V(F)$ denote the set of vertices incident to an edge in F. For a vertex set S, let $G[S]$ be the subgraph of G induced by S. We often use the symbol Δ to denote the maximum degree of a vertex in the input graph. We use $\mathsf{OPT}(G)$ to denote the optimal value of a graph G in the graph problem we are concerned with. We simply write OPT when G is clear from the context. We denote by $\mathcal{G}$ the (infinite) set consisting of all graphs. Given a universe U and two sets $A, B \subseteq U$, the notation $d_{\text{Ham}}(A, B)$ denotes the Hamming distance between the indicator vectors of A and B (with respect to U).

7.6 MINIMUM SPANNING FOREST

To get intuition about average sensitivity of algorithms, we start with the *minimum spanning forest problem.* In this problem, we are given a weighted graph $G = (V, E, w)$, where $w : E \to \mathbb{R}$ is a weight function on edges, and we want to find a forest of the minimum total weight including all the vertices.

Recall that Kruskal's algorithm (Kruskal (1956)) works as follows: Iterate over edges in the order of increasing weights, where we break ties arbitrarily. At each iteration, add the current edge to the solution if it does not form a cycle with the edges already added. The following theorem states that this simple and deterministic algorithm is stable-on-average.

Theorem 7.6.1. *The average sensitivity of Kruskal's algorithm is $O(n/m)$.*

Proof. Let $G = (V, E)$ be the input graph and T be the spanning forest obtained by running Kruskal's algorithm on G. We consider how the output changes when we remove an edge $e \in E$ from G.

If the edge e does not belong to T, clearly the output of Kruskal's algorithm on G^{-e} is also T.

Suppose that the edge e belongs to T. Let T_1 and T_2 be the two trees rooted at the endpoints of e obtained by removing e from T. If G^{-e} is not connected, that is, e is a bridge in G, then Kruskal's algorithm outputs $T_1 \cup T_2$ on G^{-e}. If G^{-e} is connected, then let e' be the first edge considered by Kruskal's algorithm among all the edges connecting $G[V(T_1)]$ and $G[V(T_2)]$, where $V(T_i)$ is the vertex set of T_i for $i \in [2]$. Then, Kruskal's algorithm outputs $T_1 \cup T_2 \cup \{e'\}$ on G^{-e}. It follows that the Hamming distance between T and the output of the algorithm on G^{-e} is at most 2.

Therefore, the average sensitivity of Kruskal's algorithm is at most

$$\frac{m - |T|}{m} \cdot 0 + \frac{|T|}{m} \cdot 2 = O\left(\frac{n}{m}\right). \qquad \square$$

We now show that Prim's algorithm (with a simple tie-breaking rule, as described in Algorithm 7.1) has high average sensitivity even on unweighted graphs. This is in contrast to the low average sensitivity of Kruskal's algorithm that we discussed earlier.

Lemma 7.6.2. *The average sensitivity of Prim's algorithm is $\Omega(m)$.*

Algorithm 7.1 Prim's Algorithm

```
Input: undirected graph $G = ([n], E)$
1: Let $T \leftarrow \{1\}$.
2: while there exists a vertex not spanned by $T$ do
3:     Let $E'$ be the set of edges with the smallest weight among all the edges in $E$
       that have exactly one endpoint in $T$.
4:     Add to $T$, an edge from $E'$ that has lexicographically smallest $T$-endpoint
       among all edges in $E'$, breaking further ties arbitrarily.
5: return Output $T$.
```

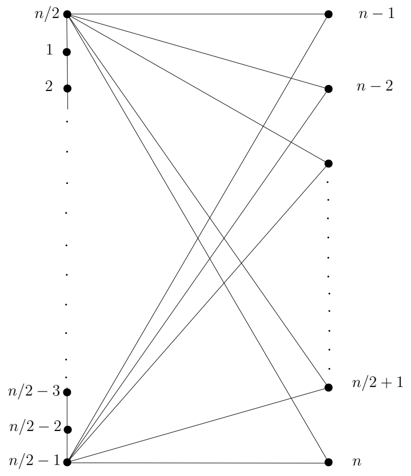

Figure 7.1: The graph family $\{G_n\}_{n\in 2\mathbb{N}}$.

Proof. Consider the graph family $\{G_n\}_{n\in 2\mathbb{N}}$ in Figure 7.1. For a large enough $n \in 2\mathbb{N}$, consider running Algorithm 7.1 on G_n. The tree T output will consist of the edges $(i, i+1)$ for all $i \in [n/2-2]$, the edges $(n/2-1, j)$ for all $j \in \{n/2+1, \ldots n\}$, and the edge $(n/2, 1)$.

If we remove an edge $(i', i'+1)$ for $i' \in [n/2-2]$ from G_n and run Algorithm 7.1 on the resulting graph, the tree, say $T_{i'}$, output will consist of all edges of the form $(i, i+1)$ for $i \in [n/2-1] \setminus \{i'\}$, all edges of the form $(n/2, j)$ for all $j \in \{n/2+1, \ldots n\}$, and the edges $(n/2+1, n/2-1)$ and $(n/2, 1)$. The Hamming distance of $T_{i'}$ from T is equal to $n/2$.

Since a uniformly random edge removed from G_n is of the form $(i, i+1)$ for

$i \in [n/2-2]$ with probability $\frac{n/2-2}{3n/2-1}$, the average sensitivity of Algorithm 7.1 is at least $\frac{n}{2} \cdot \frac{n/2-2}{3n/2-1}$, which is at least $\frac{n}{6} - 1 = \Omega(m)$ for the family $\{G_n\}_{n \in 2\mathbb{N}}$. □

7.7 GLOBAL MINIMUM CUT

For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, we define $\mathsf{cost}(G, S)$ to be the number of edges in E that cross the cut $(S, V \setminus S)$. Then in the *global minimum cut problem*, given a graph $G = (V, E)$, we want to compute a vertex set $\emptyset \subsetneq S \subsetneq V$ that minimizes $\mathsf{cost}(G, S)$.

7.7.1 Upper Bound

In this section, we show an algorithm with low average sensitivity for computing the global minimum cut problem in undirected graphs. Specifically, we show the following.

Theorem 7.7.1. *For $\varepsilon > 0$, there exists a polynomial-time algorithm for the global minimum cut problem with approximation ratio $2+\varepsilon$ and average sensitivity $n^{O(\frac{1}{(\varepsilon \cdot \mathsf{OPT})})}$.*

Let OPT be the minimum size of a cut in G. Our algorithm enumerates cuts of small size and then output a vertex set S with probability $\exp(-\alpha \cdot \mathsf{cost}(G, S))$ for a suitable α. See Algorithm 7.2 for details.

Algorithm 7.2 Stable Algorithm for Global Minimum Cut

```
Input: undirected graph $G = (V, E)$, $\varepsilon > 0$
1: Compute the value $\mathsf{OPT}$.
2: Let $\alpha \leftarrow \frac{(2+1/\varepsilon)\log n}{\mathsf{OPT}}$ denote a parameter.
3: Enumerate all cuts of size at most $(2+7\varepsilon)\mathsf{OPT} + 2\varepsilon$.
4: Sample a vertex set $S$ (from among the cuts enumerated) with probability proportional to $\exp(-\alpha \cdot \mathsf{cost}(G, S))$.
5: return $S$.
```

The approximation ratio of the Algorithm 7.2 is $2+9\varepsilon$: It clearly holds when $\mathsf{OPT} \geq 1$, and it also holds when $\mathsf{OPT} = 0$ because we only output a cut of size zero (for $\varepsilon < 1/2$). The following theorem due to Karger (Karger (1993)) directly implies that it runs in time polynomial in the input size for any constant $\varepsilon > 0$.

Theorem 7.7.2 (Karger (1993)). *Given a graph G on n vertices with the minimum cut size c and a parameter $\alpha \geq 1$, the number of cuts of size at most $\alpha \cdot c$ is at most $n^{2\alpha}$ and can be enumerated in time polynomial (in n) per cut.*

We now show that Algorithm 7.2 is stable-on-average.

Lemma 7.7.3. *The average sensitivity of Algorithm 7.2 is at most*

$$\beta(G) = \frac{n}{m} \cdot n^{(2+1/\varepsilon)/\mathsf{OPT}} \cdot ((2+7\varepsilon)\,\mathsf{OPT} + 2\varepsilon) + o(1).$$

As we have $\mathsf{OPT} \leq 2m/n$, the average sensitivity can be bounded by $n^{O(1/(\varepsilon \cdot \mathsf{OPT}))}$, and Theorem 7.7.1 follows by replacing ε with $\varepsilon/9$.

Proof. If $\mathsf{OPT} = 0$, then the claim trivially holds because the right hand size is infinity. Hence in what follows, we assume $\mathsf{OPT} \geq 1$.

Let $\mathcal{A}$ denote Algorithm 7.2. Consider an (inefficient) algorithm $\mathcal{A}'$ that on input G, outputs a cut $S \subseteq V$ (from among all the cuts in G) with probability proportional to $\exp(-\alpha \cdot \mathsf{cost}(G,S))$. For a graph $G=(V,E)$, let $\mathcal{A}(G)$ and $\mathcal{A}'(G)$ denote the output distribution of algorithms $\mathcal{A}$ and $\mathcal{A}'$ on input G, respectively. For $G=(V,E)$ and $S \subseteq V$, let $p_G(S)$ and $p'_G(S)$ be shorthands for the probabilities that S is output on input G by algorithms $\mathcal{A}$ and $\mathcal{A}'$, respectively.

We first bound the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}'(G)$ for a graph

$G = (V, E)$. To this end, we define

$$Z = \sum_{S \subseteq V : \mathsf{cost}(G,S) \le \mathsf{OPT}+b} \exp(-\alpha \cdot \mathsf{cost}(G,S)), \text{ and } Z' = \sum_{S \subseteq V} \exp(-\alpha \cdot \mathsf{cost}(G,S))$$

where $b = (1 + 7\varepsilon)\mathsf{OPT} + 2\varepsilon$. Note that $Z \le Z'$ and the quantity $\frac{Z'-Z}{Z'}$ is the total probability mass assigned by algorithm $\mathcal{A}'$ to cuts $S \subseteq V$ such that $\mathsf{cost}(G,S) > \mathsf{OPT} + b$.

Now, we start with $\mathcal{A}'(G)$. For each $S \subseteq V$ such that $\mathsf{cost}(G,S) \le \mathsf{OPT} + b$, keep at least $\frac{Z}{Z'} \cdot p'_G(S)$ mass with a cost of 0 and move a mass of at most $p'_G(S) - \frac{Z}{Z'} \cdot p'_G(S)$ at a cost of $n \cdot (p'_G(S) - \frac{Z}{Z'} \cdot p'_G(S))$. For each $S \subseteq V$ such that $\mathsf{cost}(G,S) > \mathsf{OPT} + b$, we move a mass of $p'_G(S)$ at a cost of $n \cdot p'_G(S)$. The total cost of moving masses is then equal to:

$$\begin{aligned} d_{\mathrm{EM}}\left(\mathcal{A}(G), \mathcal{A}'(G)\right) \le n \cdot & \sum_{S \subseteq V : \mathsf{cost}(G,S) \le \mathsf{OPT}+b} p'_G(S) \left(1 - \frac{Z}{Z'}\right) \\ & + n \cdot \sum_{S \subseteq V : \mathsf{cost}(G,S) > \mathsf{OPT}+b} p'_G(S) \\ & = \frac{n(Z'-Z)}{Z'} \left(\sum_{S \subseteq V : \mathsf{cost}(G,S) \le \mathsf{OPT}+b} p'_G(S) + 1 \right) \\ & \le \frac{2n(Z'-Z)}{Z'}. \end{aligned}$$

Let n_t stand for the number of cuts of cost at most $\mathsf{OPT} + t$ in G. By Karger's theorem (Theorem 7.7.2), we have that $n_t \le n^{2+2t/\mathsf{OPT}}$. Then, we have

$$\begin{aligned} \frac{Z'-Z}{Z'} & \le \sum_{t > b} \exp(-\alpha t) \cdot (n_t - n_{t-1}) \le (\exp(\alpha) - 1) \cdot \sum_{t > b} \exp(-\alpha t) n_t \\ & \le (\exp(\alpha) - 1) n^2 \cdot \sum_{t > b} n^{2t/\mathsf{OPT}} \cdot \exp(-\alpha t) \end{aligned}$$

$$\begin{aligned}
&\le (\exp(\alpha)-1)n^2 \cdot \sum_{t>b} n^{-t/(\varepsilon\cdot\mathsf{OPT})} \le (\exp(\alpha)-1)n^2 \cdot \frac{n^{-(b+1)/(\varepsilon\cdot\mathsf{OPT})}}{1-n^{-1/(\varepsilon\cdot\mathsf{OPT})}} \\
&= \left(n^{(2+1/\varepsilon)/\mathsf{OPT}}-1\right)\cdot\left(1+\frac{1}{n^{1/(\varepsilon\cdot\mathsf{OPT})}-1}\right)\cdot\frac{n^2}{n^{(b+1)/(\varepsilon\cdot\mathsf{OPT})}} \\
&\le n^{(2+1/\varepsilon)/\mathsf{OPT}}\cdot\left(1+\frac{\varepsilon n}{\log n}\right)\cdot\frac{n^2}{n^{(b+1)/(\varepsilon\cdot\mathsf{OPT})}} \\
&= O\left(\frac{\varepsilon n^{3+(2+1/\varepsilon)/\mathsf{OPT}}}{n^{(b+1)/(\varepsilon\cdot\mathsf{OPT})}}\right) = O\left(\frac{\varepsilon}{n^{4+1/\varepsilon}}\right).
\end{aligned}$$

The last inequality above follows from our choice of b. Therefore, the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}'(G)$ is $d_{\mathrm{EM}}\big(\mathcal{A}(G),\mathcal{A}'(G)\big) \le O(\frac{\varepsilon}{n^{3+1/\varepsilon}})$.

In addition, we can bound the expected size of the cut output by $\mathcal{A}'$ on G as follows. The total probability mass assigned by algorithm $\mathcal{A}'$ to cuts of size larger than $\mathsf{OPT}+b$ is equal to $\frac{Z'-Z}{Z'} = O\left(\frac{\varepsilon}{n^{4+1/\varepsilon}}\right)$. Hence, the expected size of the cut output by $\mathcal{A}'$ on G is at most $\mathsf{OPT}+b+m\cdot O(\frac{\varepsilon}{n^{4+1/\varepsilon}}) = (2+7\varepsilon)\mathsf{OPT}+2\varepsilon+O(\frac{\varepsilon m}{n^{4+1/\varepsilon}})$.

We now bound the earth mover' distance between $\mathcal{A}'(G)$ and $\mathcal{A}'(G^{-e})$ for an arbitrary edge $e \in E$. Let Z'_e denote the quantity $\sum_{S\subseteq V}\exp(-\alpha\cdot\mathsf{cost}(G^{-e},S))$. Since the cost of every cut in G^{-e} is at most the cost of the same cut in G, we have that $Z' \le Z'_e$ and therefore,

$$p'_G(S) = \frac{\exp(-\alpha\cdot\mathsf{cost}(G,S))}{Z'} \le \frac{\exp(\alpha\cdot\mathsf{cost}(G^{-e},S))}{Z'_e}\cdot\frac{Z'_e}{Z'} = p'_{G^{-e}}(S)\cdot\frac{Z'_e}{Z'}.$$

We transform $\mathcal{A}'(G)$ into $\mathcal{A}'(G^{-e})$ as follows. For each $S\subseteq V$, we leave a probability mass of at most $p'_{G^{-e}}(S)$ at S with zero cost and move a mass of $\max\{0, p'_G(S) - p'_{G^{-e}}(S)\}$ to any other point at a cost of at most $n\cdot\max\{0,p'_G(S)-p'_{G^{-e}}(S)\} \le n\cdot\left(\frac{Z'_e}{Z'}-1\right)\cdot p'_G(S)$. Hence,

$$d_{\mathrm{EM}}\left(\mathcal{A}'(G),\mathcal{A}'(G^{-e})\right) \le n\cdot\left(\frac{Z'_e}{Z'}-1\right)\cdot\sum_{S\subseteq V}p'_G(S) = n\cdot\left(\frac{Z'_e}{Z'}-1\right).$$

By the triangle inequality, the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$ can be bounded as

$$\begin{aligned}
&d_{\mathrm{EM}}\big(\mathcal{A}(G), \mathcal{A}(G^{-e})\big) \\
&\leq d_{\mathrm{EM}}\left(\mathcal{A}(G), \mathcal{A}'(G)\right) + d_{\mathrm{EM}}\left(\mathcal{A}'(G), \mathcal{A}'(G^{-e})\right) + d_{\mathrm{EM}}\left(\mathcal{A}'(G^{-e}), \mathcal{A}(G^{-e})\right) \\
&\leq n \cdot \left(\frac{Z'_e}{Z'} - 1\right) + O\left(\frac{2\varepsilon}{n^{2+1/\varepsilon}}\right).
\end{aligned}$$

Hence, the average sensitivity of $\mathcal{A}$ is bounded as:

$$\begin{aligned}
\beta(G) &= \mathbb{E}_{e\sim E}\, d_{\mathrm{EM}}\big(\mathcal{A}(G), \mathcal{A}(G^{-e})\big) \\
&\leq O\left(\frac{2\varepsilon}{n^{3+1/\varepsilon}}\right) + n \cdot \mathbb{E}_{e\sim E}\left(\frac{Z'_e}{Z'} - 1\right) \\
&= O\left(\frac{2\varepsilon}{n^{3+1/\varepsilon}}\right) + \frac{n}{mZ'}\sum_{e\sim E}(Z'_e - Z').
\end{aligned}$$

The second term in the above expression can be bounded as:

$$\begin{aligned}
&\frac{n}{mZ'}\sum_{e\in E}(Z'_e - Z') \\
&= \frac{n}{mZ'}\sum_{e\in E}\sum_{S\subseteq V: e \text{ crosses } S} \exp(-\alpha\cdot\mathsf{cost}(G^{-e}, S)) - \exp(-\alpha\cdot\mathsf{cost}(G, S)) \\
&= \frac{n(\exp(\alpha)-1)}{mZ'}\sum_{e\in E}\sum_{S\subseteq V: e \text{ crosses } S} \exp(-\alpha\cdot\mathsf{cost}(G, S)) \\
&= \frac{n(\exp(\alpha)-1)}{m}\sum_{S\subseteq V}\mathsf{cost}(G, S)\cdot\frac{\exp(-\alpha\cdot\mathsf{cost}(G, S))}{Z'}.
\end{aligned}$$

The summation in the expression above is equal to the expected size of the cut output by algorithm $\mathcal{A}'$ on input G. We argued that it is at most $(2+7\varepsilon)\mathsf{OPT} + 2\varepsilon +$

$O(\frac{\varepsilon m}{n^{4+1/\varepsilon}})$. Hence, the average sensitivity of $\mathcal{A}$ is at most

$$\begin{aligned}
&\frac{n}{m} \cdot n^{(2+1/\varepsilon)/\mathsf{OPT}} \cdot ((2+7\varepsilon)\,\mathsf{OPT} + 2\varepsilon) + O\left(\frac{\varepsilon n^{(2+1/\varepsilon)/\mathsf{OPT}} + 2}{n^{3+1/\varepsilon}}\right) \\
&= \frac{n}{m} \cdot n^{(2+1/\varepsilon)/\mathsf{OPT}} \cdot ((2+7\varepsilon)\,\mathsf{OPT} + 2\varepsilon) + o(1)
\end{aligned}$$

as $\mathsf{OPT} \geq 1$. □

7.8 MAXIMUM MATCHING

A vertex-disjoint set of edges is called a *matching*. In the maximum matching problem, given a graph, we want to find a matching of the maximum size. In this section, we describe different algorithms with low average sensitivity that approximate the maximum matching in a graph.

7.8.1 Lexicographically Smallest Matching

In this section, we describe an algorithm that computes a maximum matching in a graph with average sensitivity at most OPT^2/m and prove Theorem 7.8.1, where OPT is the maximum size of a matching.

First, we define some ordering among vertex pairs. Then, we can naturally define the lexicographical order among matchings by regarding a matching as a sorted sequence of vertex pairs. Then, our algorithm simply outputs the lexicographically smallest matching. Note that this can be done in polynomial time using Edmonds' algorithm (Edmonds (1965)).

Theorem 7.8.1. *Let $\mathcal{A}$ be the algorithm that outputs the lexicographically smallest maximum matching. Then, the average sensitivity of $\mathcal{A}$ is at most OPT^2/m, where OPT is the maximum size of a matching.*

Proof. For a graph $G = (V, E)$, let $M(G) \subseteq E$ be its lexicographically smallest maximum matching. As long as $e \notin M$, we have $M(G) = M(G^{-e})$. Hence, the average sensitivity of the algorithm is at most

$$\frac{\mathsf{OPT}}{m} \cdot \mathsf{OPT} + \left(1 - \frac{\mathsf{OPT}}{m}\right) \cdot 0 = \frac{\mathsf{OPT}^2}{m}. \qquad \square$$

Remark 7.8.2. *Consider the path graph $P_n = ([n], E)$, where $E = \{(i, i+1) : i \in [n-1]\}$. The average sensitivity of the above algorithm on P_n is $\Omega(\frac{\mathsf{OPT}^2}{m})$. Hence the above analysis of the average sensitivity is tight.*

7.8.2 Greedy Matching Algorithm

In this section, we describe an algorithm (based on a randomized greedy maximal matching algorithm) with average sensitivity $\tilde{O}\left(\mathsf{OPT}^{3/4}\right)$ and approximation ratio $1/2 - o(1)$ for the maximum matching problem and prove Theorem 7.8.9.

In Theorem 7.8.3, we prove that the basic randomized greedy maximal matching algorithm has sensitivity $O(\Delta)$, where Δ is the maximum degree of the input graph.

Theorem 7.8.4 shows how to transform the randomized greedy algorithm to another algorithm whose average sensitivity does not depend on the maximum degree, albeit at the cost of slightly worsening the approximation guarantee. In particular, Theorem 7.8.8 shows that a $(1/2-\varepsilon)$-approximation algorithm for maximum matching with average sensitivity $O\left(\frac{\varepsilon}{1-\varepsilon}\log n + \frac{m^3}{\varepsilon^3 \mathsf{OPT}^3}\right)$ is obtained by applying Theorem 7.8.4 to the randomized greedy maximal matching.

Finally, we combine the matching algorithm guaranteed by Theorem 7.8.8 with the matching algorithm guaranteed by Theorem 7.8.1 using the parallel composition property (Theorem 7.11.2) of stable-on-average algorithms and obtain Theorem 7.8.9.

7.8.2.1 Average sensitivity of the greedy algorithm in terms of the maximum degree

In this section, we describe the average sensitivity guarantee of the randomized greedy algorithm described in Algorithm 7.3. It is evident that Algorithm 7.3 runs in polyno-

Algorithm 7.3 Randomized Greedy Algorithm

```
Input: undirected unweighted graph $G = (V, E)$
1: Sample a uniformly random ordering $\pi$ of edges in $E$.
2: Set $M \leftarrow \emptyset$.
3: Consider edges one by one according to $\pi$ and add an edge $(u, v)$ to $M$ only if
   both $u$ and $v$ are unmatched in $M$.
4: return $M$.
```

mial time and that the matching it outputs has size at least $\frac{1}{2}$ the size of a maximum matching in the input graph.

Theorem 7.8.3. *For every undirected unweighted graph G, the average sensitivity of Algorithm 7.3 is $\beta(G) \leq \frac{1}{2} + \Delta$, where Δ is the maximum degree of G.*

Proof. Consider a graph $G = (V, E)$. Let Δ be the maximum degree of G. For an edge $e \in E$, let G^{-e} denote the graph obtained by removing e from G. Let $M(G)$ denote the matching output by Algorithm 7.3 on input G. Yoshida et al. (Yoshida et al., 2012, Theorem 2.1) show that the presence of a uniformly random edge e in $M(G)$ depends on at most $\frac{1}{2} + \Delta$ edges in expectation, where the expectation is taken over both the randomness of the algorithm and the randomness in selecting the edge e. By applying Theorem 7.2.5 to this statement, we can see that the average sensitivity of Algorithm 7.3 is $\beta(G) \leq \frac{1}{2} + \Delta$, where Δ is the maximum degree of G. $\square$

7.8.2.2 Stable-on-average thresholding transformation

In this section, we show a transformation from matching algorithms whose average sensitivity is a function of the maximum degree to matching algorithms whose average sensitivity does not depend on the maximum degree. This is done by adding to the algorithm, a preprocessing step that removes vertices from the input graph, where the removed vertices have degree at least an appropriate random threshold. Such a transformation helps us to design stable-on-average algorithms for graphs with unbounded degree. Let $\mathsf{Lap}(\mu, \phi)$ denote the Laplace distribution with a location parameter μ and a scale parameter ϕ.

Theorem 7.8.4. *Let $\mathcal{A}'$ be a randomized algorithm for the maximum matching problem such that the size of the matching output by $\mathcal{A}'$ on a graph G is always at least $a \cdot \mathsf{OPT}$ for some $a \geq 0$. In addition, assume that there exists an oracle $\mathcal{O}$ satisfying the following:*

- *when given access to a graph $G = (V, E)$ and query $e \in E$, the oracle generates a random string $\pi \in \{0,1\}^r$ and outputs whether e is contained in the matching output by $\mathcal{A}'$ on G with π as its random string, and*
- *the oracle $\mathcal{O}$ makes at most $q(\Delta)$ queries to G in expectation, where Δ is the maximum degree of G and the expectation is taken over the random coins of $\mathcal{A}'$ and a uniformly random query $e \in E$.*

Let $\delta > 0$ and τ be a non-negative function on graphs. Then, there exists an algorithm $\mathcal{A}$ for the maximum matching problem with average sensitivity

$$\beta(G) \leq O\left(\frac{K_G}{\delta(\tau(G) - K_G)} + \exp\left(-\frac{1}{\delta}\right)\right) \cdot \mathsf{OPT} + \mathbb{E}_L\left[(2L-2)^2 q(L)\right],$$

where L is a random variable distributed as $\mathsf{Lap}(\tau(G), \delta\tau(G))$ and K_G is shorthand for $\max_{e \in E(G)} |\tau(G) - \tau(G^{-e})|$. Moreover, the expected size of the matching output by $\mathcal{A}$ is at least

$$a \cdot \mathsf{OPT} - \frac{am}{(1 - \delta \ln(\mathsf{OPT}/2)) \cdot \tau(G)} - a.$$

The following fact will be useful in the proof of Theorem 7.8.4.

Proposition 7.8.5. *Let L be a random variable distributed as* $\mathsf{Lap}(\mu, \phi)$. *Then,* $\Pr[L < (1-\varepsilon)\mu] \le \exp(-\varepsilon\mu/\phi)/2$. *Similarly,* $\Pr[L > (1+\varepsilon)\mu] \le \exp(-\varepsilon\mu/\phi)/2$.

Proof of Theorem 7.8.4. The algorithm $\mathcal{A}$ is given below.

Algorithm $\mathcal{A}$: On input $G = (V, E)$,

1. Sample a random variable L according to the distribution $\mathsf{Lap}(\tau(G), \delta\tau(G))$.
2. Let $[G]_L$ be the graph obtained after removing from G all vertices of degree at least L.
3. Run $\mathcal{A}'$ on $[G]_L$.

We first bound the average sensitivity of $\mathcal{A}$. We can think of $\mathcal{A}$ as being sequentially composed of two algorithms, where the first algorithm takes in a graph $G = (V, E)$ and outputs a number $L \sim \mathsf{Lap}(\tau(G), \delta\tau(G))$. The second algorithm takes both L and G and runs $\mathcal{A}'$ on $[G]_L$.
Let L_e for $e \in E$ denote a Laplace random variable $\mathsf{Lap}(\tau(G^{-e}), \delta \cdot \tau(G^{-e}))$. Using Theorem 7.2.2, we get that the average sensitivity of $\mathcal{A}$ is bounded by

$$\mathsf{OPT} \cdot \mathbb{E}_{e\sim E}\left[d_{\mathrm{TV}}(L, L_e)\right] + \mathbb{E}_L\left[\mathbb{E}_{e\sim E}\left[d_{\mathrm{EM}}(\mathcal{A}'([G]_L), \mathcal{A}'([G^{-e}]_L))\right]\right].$$

Claim 7.8.6. *For $x \in \mathbb{R}$,* $\mathbb{E}_{e\sim E}\left[d_{\mathrm{EM}}\big(\mathcal{A}'([G]_x), \mathcal{A}'([G^{-e}]_x)\big)\right] \le (2x-2)^2 q(x)$.

Proof. Fix $x \in \mathbb{R}$. In order to bound the term $\mathbb{E}_{e\sim E}\left[d_{\mathrm{EM}}\big(\mathcal{A}'([G]_x), \mathcal{A}'([G^{-e}]_x)\big)\right]$, consider the following algorithm $\mathcal{A}'_x$. On input $G = (V, E)$, the algorithm $\mathcal{A}'_x$ first removes every vertex of degree at least x from G and then runs $\mathcal{A}'$ on the resulting graph. Hence, the quantity $\mathbb{E}_{e\sim E}\left[d_{\mathrm{EM}}\big(\mathcal{A}'([G]_x), \mathcal{A}'([G^{-e}]_x)\big)\right]$ denotes the average sensitivity of $\mathcal{A}'_x$.

In order to bound the average sensitivity of $\mathcal{A}'_x$, construct an oracle $\mathcal{O}_x$ as follows. $\mathcal{O}_x$ when given access to a graph $G = (V, E)$ and input e sampled uniformly at random from E, does the following. It first checks whether at least one of the endpoints of e has degree at least x. If so, it returns that e does not belong to the solution obtained by running $\mathcal{A}'_x$ on G. Otherwise, it runs $\mathcal{O}$ with access to $[G]_x$ and e as input and outputs the answer of $\mathcal{O}$.

We can analyze the query complexity of $\mathcal{O}_x$ as follows. Call an edge $e \in E$ *alive* if both the endpoints of e have degree less than x. Otherwise, e is *dead*.

The oracle $\mathcal{O}_x$ can check whether an edge $e = (u, v)$ is alive or not by querying at most $2x - 2$ edges incident to e. In particular $\mathcal{O}_x$ examines the neighbors of u and v one by one, and, as soon $\mathcal{O}_x$ encounters $x - 1$ distinct neighbors (excluding u or v themselves) for either u or v, $\mathcal{O}_x$ can declare e to be a dead edge.

If the edge $e \in E$ input to $\mathcal{O}_x$ is a dead edge, therefore, $\mathcal{O}_x$ queries at most $2x - 2$ edges and returns that e cannot be part of a solution to running $\mathcal{A}'_x$ on G.

If the input edge $e \in E$ is alive, then we know that it is a uniformly random alive edge. By the guarantee on $\mathcal{O}$, we then know that $\mathcal{O}$ makes at most $q(x)$ queries to the alive edges in expectation over the randomness of $\mathcal{A}'$ and the choice of the input alive edge, since the maximum degree of $[G]_x$ is at most x. In order for the oracle $\mathcal{O}_x$ to simulate oracle access to $[G]_x$ for the oracle $\mathcal{O}$, for each alive edge e queried by $\mathcal{O}$, the oracle $\mathcal{O}_x$ has to query each edge incident to e in G and determine which among these are alive. Since e is alive, both endpoints of e have degrees less than x. Hence, $\mathcal{O}_x$ need only check whether at most $2x - 2$ edges incident to e are alive or not. This can be done by querying $(2x - 2)^2$ edges in E in total.

Combining all of the above, the expected query complexity of $\mathcal{O}_x$ is at most $(2x - 2)^2 q(x)$, where the expectation is taken over the edges of $e \in E$ and the randomness in $\mathcal{A}_x$.

Therefore, by Theorem 7.2.5, we get that the average sensitivity of algorithm $\mathcal{A}_x$ is bounded by $(2x-2)^2 q(x)$. $\square$

We now bound the quantity $\mathbb{E}_{e\sim E}\left[d_{\mathrm{TV}}(L, L_e)\right]$.

Claim 7.8.7. *For any $e \in E$, we have*

$$d_{\mathrm{TV}}(L, L_e) \leq O\left(\frac{K}{\delta(\tau - K)} + \exp\left(-\frac{1}{\delta}\right)\right).$$

Proof. Let $f_L, f_{L_e} \colon \mathbb{R} \to \mathbb{R}$ be the probability density functions of the Laplace random variables L and L_e, respectively. Let $\tau = \tau(G)$, $\tau_e = \tau(G_e)$, and $K = K_G$. Then

$$\begin{aligned}\frac{f_L(x)}{f_{L_e}(x)} &= \frac{\frac{1}{2\delta\tau}\exp\left(-\frac{|x-\tau|}{\delta\cdot\tau}\right)}{\frac{1}{2\delta\cdot\tau_e}\exp\left(-\frac{|x-\tau_e|}{\delta\cdot\tau_e}\right)} = \frac{\tau_e}{\tau}\exp\left(\frac{|x-\tau_e|}{\delta\cdot\tau_e} - \frac{|x-\tau|}{\delta\cdot\tau}\right) \\ &= \left(1 - \frac{\tau-\tau_e}{\tau}\right)\exp\left(\frac{\tau|x-\tau_e| - \tau_e|x-\tau|}{\delta\tau\tau_e}\right).\end{aligned}$$

A direct calculation shows that for $0 \leq x \leq 2\max\{\tau, \tau_e\}$, we have

$$\left(1 - \frac{K}{\tau}\right)\exp\left(\frac{-2K}{\delta(\tau-K)}\right) \leq \frac{f_L(x)}{f_{L_e}(x)} \leq \left(1 + \frac{K}{\tau}\right)\exp\left(\frac{2K}{\delta(\tau-K)}\right).$$

This implies that for all $S \subseteq [0, 2\max\{\tau, \tau_e\}]$,

$$\begin{aligned}\left(1 - \frac{K}{\tau}\right)\exp\left(\frac{-2K}{\delta(\tau-K)}\right) - 1 &\leq \Pr[L \in S] - \Pr[L_e \in S] \\ \Pr[L \in S] - \Pr[L_e \in S] &\leq \left(1 + \frac{K}{\tau}\right)\exp\left(\frac{2K}{\delta(\tau-K)}\right) - 1.\end{aligned}$$

By Proposition 7.8.5, the probability that L (and L_e as well) falls in the range $[-\infty, 0] \cup [2\max\{\tau, \tau_e\}, \infty]$ is bounded by $\exp(-1/\delta)$. Hence, total variation distance between L and L_e is

$$\begin{aligned}
&\left(1+\frac{K}{\tau}\right)\exp\left(\frac{2K}{\delta(\tau-K)}\right)-\left(1-\frac{K}{\tau}\right)\exp\left(\frac{-2K}{\delta(\tau-K)}\right)+2\exp\left(-\frac{1}{\delta}\right)\\
&=\left(1+\frac{K}{\tau}\right)\left(1+\frac{2K}{\delta(\tau-K)}+O\left(\frac{K^2}{\delta^2(\tau-K)^2}\right)\right)\\
&\quad-\left(1-\frac{K}{\tau}\right)\left(1-\frac{2K}{\delta(\tau-K)}-O\left(\frac{K^2}{\delta^2(\tau-K)^2}\right)\right)+2\exp\left(-\frac{1}{\delta}\right)\\
&=\frac{2K}{\tau}+\frac{4K}{\delta(\tau-K)}+O\left(\frac{K^2}{\delta^2(\tau-K)^2}\right)+2\exp\left(-\frac{1}{\delta}\right)\\
&\leq\frac{6K}{\delta(\tau-K)}+2\exp\left(-\frac{1}{\delta}\right)+O\left(\frac{K^2}{\delta^2(\tau-K)^2}\right).\\
&=O\left(\frac{K}{\delta(\tau-K)}+\exp\left(-\frac{1}{\delta}\right)\right). \qquad \square
\end{aligned}$$

Therefore, the average sensitivity of $\mathcal{A}$ is bounded as

$$\begin{aligned}
\beta(G)&=\mathbb{E}_{e\sim E}d_{\mathrm{EM}}\big(\mathcal{A}(G),\mathcal{A}(G^{-e})\big)\\
&\leq O\left(\frac{K}{\delta(\tau-K)}+\exp\left(-\frac{1}{\delta}\right)\right)\cdot\mathsf{OPT}+\mathbb{E}_L\left[(2x-2)^2q(x)\right].
\end{aligned}$$

We now bound the approximation guarantee of $\mathcal{A}$. By Proposition 7.8.5,

$$\Pr\left[L<\left(1-\delta\ln\left(\frac{\mathsf{OPT}}{2}\right)\right)\cdot\tau(G)\right]\leq\frac{1}{\mathsf{OPT}}.$$

Therefore, with probability at least $1-1/\mathsf{OPT}$, only those vertices with degree at least $(1-\delta\ln(\mathsf{OPT}/2))\cdot\tau(G)$ are removed from G. The number of such vertices is at most $\frac{m}{(1-\delta\ln(\mathsf{OPT}/2))\cdot\tau(G)}$. Therefore, with probability at least $1-1/\mathsf{OPT}$, the size of a maximum matching in the resulting graph is at most $\frac{m}{(1-\delta\ln(\mathsf{OPT}/2))\cdot\tau(G)}$ smaller than that of G. With probability at most $1/\mathsf{OPT}$, the size of a maximum matching in the resulting instance could be smaller by an additive term of at most OPT. Hence, the expected size of a maximum matching in the new instance is at least

$$\mathsf{OPT} - \frac{m}{(1 - \delta \ln(\mathsf{OPT}(G)/2)) \cdot \tau(G)} - 1.$$

The statement on approximation guarantee follows. $\square$

7.8.2.3 Average sensitivity of the greedy algorithm with thresholding

In this section, we apply Theorem 7.8.4 to Algorithm 7.3 and analyze the average sensitivity of the resulting algorithm. We show the following.

Theorem 7.8.8. *Let $\varepsilon \in (0,1)$ be a parameter. There exists an algorithm $\mathcal{A}_\varepsilon$ with approximation ratio $1/2 - \varepsilon$ and sensitivity $O\left(\frac{\varepsilon}{1-\varepsilon} \cdot \log n + \frac{m^3}{\varepsilon^3 \mathsf{OPT}(G)^3}\right)$.*

Proof. The algorithm guaranteed by the theorem statement is as follows.

Algorithm $\mathcal{A}_\varepsilon$: On input $G = (V, E)$,

1. Compute OPT.
2. If $\mathsf{OPT} \leq \frac{1}{\varepsilon} + 1$ or $m \leq \frac{1}{2\varepsilon}$, then output an arbitrary maximum matching.
3. Otherwise, run the algorithm obtained by applying Theorem 7.8.4 to Algorithm 7.3 with the setting $\tau := \tau(G) = \frac{m}{\varepsilon' \mathsf{OPT}}$ and $\delta := \frac{1}{2 \ln n}$, where $\varepsilon' = \varepsilon - \frac{1}{2\mathsf{OPT}}$.

Approximation guarantee: If $\mathsf{OPT} \leq \frac{1}{\varepsilon} + 1$ or $m \leq \frac{1}{2\varepsilon}$, the approximation guarantee is clear. Otherwise, since Algorithm 7.3 outputs a maximal matching whose size is always at least $\frac{\mathsf{OPT}}{2}$, the size of the matching output by $\mathcal{A}_\varepsilon$ is at least $\frac{\mathsf{OPT}}{2} - \frac{\frac{\varepsilon'}{2}\mathsf{OPT}}{1 - \frac{\ln(\mathsf{OPT}/2)}{2 \ln n}} - \frac{1}{2}$, which is at least

$$\frac{\mathsf{OPT}}{2} - \varepsilon \cdot \mathsf{OPT}$$

by the setting of ε' and the fact that $\frac{\ln(\mathsf{OPT}/2)}{2 \ln n} \leq \frac{1}{2}$.

Average sensitivity: If $\mathsf{OPT} \leq \frac{1}{\varepsilon}+1$ or $m \leq \frac{1}{2\varepsilon}$, the average sensitivity of $\mathcal{A}_\varepsilon$ is bounded by $O(\frac{1}{\varepsilon})$, since the size of maximum matching in G is small and it can decrease only by at most 1 by the removal of an edge.

We now analyze the average sensitivity of $\mathcal{A}_\varepsilon$ for the case that $\mathsf{OPT} > \frac{1}{\varepsilon}+1$. We use the shorthand OPT_e to denote $\mathsf{OPT}(G^{-e})$ for $e \in E$. Note that

$$\begin{aligned}
K_G &= 2\max_{e\in E}\left|\frac{m}{2(\varepsilon\cdot\mathsf{OPT})-1} - \frac{m-1}{2\varepsilon\cdot\mathsf{OPT}_e - 1}\right| \\
&\leq 2\max\left\{\frac{m}{2\varepsilon\cdot\mathsf{OPT}-1} - \frac{m-1}{2\varepsilon\cdot\mathsf{OPT}-1}, \frac{m-1}{2\varepsilon(\mathsf{OPT}-1)-1} - \frac{m}{2\varepsilon\cdot\mathsf{OPT}-1}\right\} \\
&= 2\max\left\{\frac{1}{2\varepsilon\cdot\mathsf{OPT}-1}, \frac{2\varepsilon(m-\mathsf{OPT})+1}{(2\varepsilon(\mathsf{OPT}-1)-1)\cdot(2\varepsilon\cdot\mathsf{OPT}-1)}\right\} \\
&= \frac{2}{2\varepsilon\cdot\mathsf{OPT}-1}\max\left\{1, \frac{2\varepsilon(m-\mathsf{OPT})+1}{2\varepsilon(\mathsf{OPT}-1)-1}\right\}.
\end{aligned}$$

The second inequality above uses the fact that for numbers $a, b, c \geq 0$ such that $a \leq b$ and $c(b-1)-1 \geq 0$, we have that $\frac{a}{cb-1} \leq \frac{a-1}{c(b-1)-1}$.

From the statement of Theorem 7.8.3, we can see that $q(x) \leq \frac{1}{2} + x$ when $x > 0$ and $q(x) = 0$ otherwise. Therefore, we can see that the average sensitivity of the algorithm resulting from applying Theorem 7.8.4 to Algorithm 7.3 is bounded as:

$$O\left(\frac{K_G}{\delta(\tau-K_G)} + \exp\left(-\frac{1}{\delta}\right)\right)\cdot\mathsf{OPT} \tag{7.5}$$

$$+\int_0^\infty (2x-2)^2\cdot\left(\frac{1}{2}+x\right)\cdot\frac{1}{2\delta\cdot\tau}\cdot\exp\left(-\frac{|x-\tau|}{\delta\cdot\tau}\right)\mathrm{d}x. \tag{7.6}$$

Since $\frac{2\varepsilon(m-\mathsf{OPT})+1}{2\varepsilon(\mathsf{OPT}-1)-1}$ is a nonincreasing function of OPT and $\mathsf{OPT} \geq \frac{1}{\varepsilon}+1$, we have that

$$\frac{2\varepsilon(m-\mathsf{OPT})+1}{2\varepsilon(\mathsf{OPT}-1)-1} \leq 2\varepsilon m.$$

Hence, $K_G \leq \frac{2}{2\varepsilon\cdot\mathsf{OPT}-1}\max\{1, 2\varepsilon m\} = \frac{4\varepsilon m}{2\varepsilon\cdot\mathsf{OPT}-1}$, since $m > \frac{1}{2\varepsilon}$ and therefore, we have that $\tau - K_G \geq \tau(1-2\varepsilon)$. Hence, the first term above can be upper bounded by

$$\begin{aligned}
&O\left(\frac{K_G}{\delta\tau(1-2\varepsilon)} + \exp\left(-\frac{1}{\delta}\right)\right)\cdot \mathsf{OPT}\\
&= O\left(\frac{4\varepsilon m}{2\varepsilon\cdot\mathsf{OPT}-1}\cdot\frac{1}{1-2\varepsilon}\cdot\frac{2\ln n}{\frac{2m}{2\varepsilon\cdot\mathsf{OPT}-1}} + \frac{\mathsf{OPT}}{n^2}\right)\\
&= O\left(\frac{\varepsilon}{1-\varepsilon}\cdot\log n\right).
\end{aligned}$$

The second term of (7.6) can be upper bounded by

$$\begin{aligned}
&\int_0^\infty (2x)^2\cdot\left(\frac{1}{2}+x\right)\cdot\frac{1}{2\delta\tau}\cdot\exp\left(-\frac{|x-\tau|}{\delta\tau}\right)\mathrm{d}x\\
&= \int_0^\infty \frac{x^2}{\delta\tau}\cdot\exp\left(-\frac{|x-\tau|}{\delta\tau}\right)\mathrm{d}x + \int_0^\infty \frac{2x^3}{\delta\tau}\cdot\exp\left(-\frac{|x-\tau|}{\delta\tau}\right)\mathrm{d}x.
\end{aligned}$$

Let I_1 and I_2 denote the first and second terms above. The term I_1 can be evaluated as:

$$\begin{aligned}
I_1 &= \int_0^\tau \frac{x^2}{\delta\tau}\cdot\exp\left(-\frac{\tau-x}{\delta\tau}\right)\mathrm{d}x + \int_\tau^\infty \frac{x^2}{\delta\tau}\cdot\exp\left(-\frac{x-\tau}{\delta\tau}\right)\mathrm{d}x\\
&= \left(1-2\delta+2\delta^2-2\exp\left(-\frac{1}{\delta}\right)\delta^2\right)\tau^2 + (1+2\delta+2\delta^2)\tau^2\\
&= \left(2+4\delta^2-2\exp\left(-\frac{1}{\delta}\right)\delta^2\right)\tau^2 = O\left(\frac{m^2}{\varepsilon^2\mathsf{OPT}^2}\right).
\end{aligned}$$

The term I_2 can be evaluated as:

$$\begin{aligned}
I_2 &= \int_0^\tau \frac{2x^3}{\delta\tau}\cdot\exp\left(-\frac{\tau-x}{\delta\tau}\right)\mathrm{d}x + \int_\tau^\infty \frac{2x^3}{\delta\tau}\cdot\exp\left(-\frac{x-\tau}{\delta\tau}\right)\mathrm{d}x\\
&= 2\left(1-3\delta+6\delta^2-6\delta^3+6\exp\left(-\frac{1}{\delta}\right)\delta^3\right)\tau^3 + 2(1+3\delta+6\delta^2+6\delta^3)\tau^3
\end{aligned}$$

$$= 2\left(2 + 12\delta^2 + 6\exp\left(-\frac{1}{\delta}\right)\delta^3\right)\tau^3 = O\left(\frac{m^3}{\varepsilon^3 \mathsf{OPT}^3}\right).$$

Hence, the average sensitivity of the algorithm obtained can be bounded by:

$$\begin{aligned}\beta(G) &= \max\left\{O\left(\frac{1}{\varepsilon}\right), O\left(\frac{\varepsilon}{1-2\varepsilon}\cdot\log n\right) + O\left(\frac{m^2}{\varepsilon^2\mathsf{OPT}^2}\right) + O\left(\frac{m^3}{\varepsilon^3\mathsf{OPT}^3}\right)\right\}\\ &= O\left(\frac{\varepsilon\log n}{1-\varepsilon} + \frac{m^3}{\varepsilon^3\mathsf{OPT}^3}\right).\end{aligned}$$ □

7.8.2.4 Average sensitivity of a combined matching algorithm

In this section, we combine the algorithms guaranteed by Theorems 7.8.1 and 7.8.8 in order to get a matching algorithm with improved sensitivity.

Theorem 7.8.9. *Let $\varepsilon \in (0, \frac{1}{2})$ be a parameter. There exists an algorithm for the maximum matching problem with approximation ratio $1/2 - \varepsilon$ and sensitivity*

$$O\left(\mathsf{OPT}^{3/4}\left(\varepsilon^{1/4}\log^{1/4} n + \frac{1}{\varepsilon^{3/4}}\right)\right).$$

In particular when $\varepsilon = 1/\log^{1/4} n$, the average sensitivity is $O(\mathsf{OPT}^{3/4}\log^{3/16} n)$.

Proof. The algorithm guaranteed by the theorem is given as Algorithm 7.4. The bounds on approximation guarantee and average sensitivity are both straightforward when $\mathsf{OPT} < 5$ or $m < 4$.

The approximation guarantee in the case when $\mathsf{OPT} \geq 5$ and $m \geq 6$ is also straightforward since Algorithm 7.4 is simply a distribution over algorithms guaranteed by Theorem 7.8.1 and Theorem 7.8.8.

We now bound the average sensitivity of Algorithm 7.4 when $\mathsf{OPT} \geq 5$ and $m \geq 6$. Let $\rho(G)$ denote the probability $\frac{g(G)}{f(G)+g(G)}$. By Theorem 7.11.2, the average sensitivity

Algorithm 7.4 Combined Algorithm to $(\frac{1}{2}-\varepsilon)$-Approximate Maximum Matching

```
Input: undirected unweighted graph $G=(V,E)$ Compute $\mathsf{OPT}$.
1: if $\mathsf{OPT} < 5$ or $m < 6$ then
2:     return an arbitrary maximum matching in $G$.
3: else
4:     Let $f(G) \leftarrow \frac{\mathsf{OPT}^2}{m}$ and $g(G) \leftarrow \frac{\varepsilon}{(1-\varepsilon)} \cdot \log n + \frac{m^3}{\varepsilon^3 \mathsf{OPT}^3}$.
5:     Run the algorithm given by Theorem 7.8.1 with probability $\frac{g(G)}{f(G)+g(G)}$ and run
       the algorithm given by Theorem 7.8.8 with the remaining probability.
```

is at most

$$\frac{O(f(G)) \cdot g(G) + O(g(G)) \cdot f(G)}{f(G)+g(G)} + 2\mathsf{OPT} \cdot \mathbb{E}_{e\sim E}\left[|\rho(G) - \rho(G^{-e})|\right]. \tag{7.7}$$

We first bound the quantity $\mathbb{E}_{e\sim E}\left[|\rho(G) - \rho(G^{-e})|\right]$.

Claim 7.8.10. *For every graph $G=(V,E)$ such that $\mathsf{OPT} \geq 5$, we have, for every $e \in E$,*

$$g(G) \cdot \left(1 - \frac{3}{m}\right) \leq g(G-e) \leq g(G) \cdot \left(1 + \frac{4}{\mathsf{OPT}-1}\right).$$

Proof. We first prove the upper bound. We know that

$$\begin{aligned}
\frac{g(G^{-e})}{g(G)} &\leq \left(1 + \frac{\left(\frac{m-1}{\varepsilon(\mathsf{OPT}-1)}\right)^3 - \left(\frac{m}{(\varepsilon\cdot\mathsf{OPT})}\right)^3}{\frac{\varepsilon \log n}{(1-\varepsilon)} + \frac{m^3}{\varepsilon^3\mathsf{OPT}^3}}\right) \\
&\leq \left(1 + \frac{\left(\frac{m-1}{\varepsilon(\mathsf{OPT}-1)}\right)^3 - \left(\frac{m}{(\varepsilon\cdot\mathsf{OPT})}\right)^3}{\frac{m^3}{\varepsilon^3\mathsf{OPT}^3}}\right) \\
&= \left(1 - \frac{1}{m}\right)^3 \cdot \left(1 + \frac{1}{\mathsf{OPT}-1}\right)^3 \\
&\leq \left(1 + \frac{4}{\mathsf{OPT}-1}\right).
\end{aligned}$$

Note that the second-to-last inequality holds whenever $\mathsf{OPT} \geq 5$ and $m \geq 3$.

For the lower bound,

$$\begin{aligned}\frac{g(G^{-e})}{g(G)} &\geq \left(1-\frac{\left(\frac{m}{(\varepsilon\cdot\mathsf{OPT})}\right)^3-\left(\frac{m-1}{\varepsilon\cdot\mathsf{OPT}}\right)^3}{\frac{\varepsilon\log n}{(1-\varepsilon)}+\frac{m^3}{\varepsilon^3\mathsf{OPT}^3}}\right)\\ &\geq \left(1-\frac{\left(\frac{m}{(\varepsilon\cdot\mathsf{OPT})}\right)^3-\left(\frac{m-1}{\varepsilon\cdot\mathsf{OPT}}\right)^3}{\frac{m^3}{\varepsilon^3\mathsf{OPT}^3}}\right)\\ &= \left(1-\frac{1}{m}\right)^3 \geq 1-\frac{3}{m}.\end{aligned}$$

□

Claim 7.8.11. *For every graphs $G=(V,E)$ and every $e\in E$,*

$$f(G)\cdot\left(1-\frac{2}{\mathsf{OPT}}\right)\leq f(G-e)\leq f(G)\cdot\left(1+\frac{1}{m-1}\right).$$

Proof. To prove the upper bound,

$$\frac{f(G^{-e})}{f(G)}\leq\left(\frac{m}{m-1}\right)=\left(1+\frac{1}{m-1}\right).$$

For the lower bound,

$$\frac{f(G^{-e})}{f(G)}\geq\left(\frac{\mathsf{OPT}-1}{\mathsf{OPT}}\right)^2\cdot\left(\frac{m}{m-1}\right)^2\geq\left(\frac{\mathsf{OPT}-1}{\mathsf{OPT}}\right)^2\geq 1-\frac{2}{\mathsf{OPT}}.$$

□

Note that $\left(1-\frac{2}{\mathsf{OPT}}\right)^{-1}\leq 1+\frac{4}{\mathsf{OPT}}$ and $\left(1-\frac{3}{m}\right)^{-1}\leq 1+\frac{6}{m}$ for $\mathsf{OPT}\geq 4$ and $m\geq 6$. We also have $\left(1+\frac{4}{\mathsf{OPT}-1}\right)^{-1}\geq 1-\frac{4}{\mathsf{OPT}-1}$ and $\left(1+\frac{1}{m-1}\right)^{-1}\geq 1-\frac{1}{m-1}$ for $\mathsf{OPT}\geq 5$ and $m\geq 2$.

Combining all of the above,

$$\rho(G^{-e})=\frac{g(G^{-e})}{f(G^{-e})+g(G^{-e})}$$

$$\begin{aligned}
&\leq \frac{g(G)\cdot\left(1+\frac{4}{\mathsf{OPT}-1}\right)}{(f(G)+g(G))\cdot\min\left\{1-\frac{3}{m},1-\frac{2}{\mathsf{OPT}}\right\}}\\
&\leq \rho(G)\cdot\left(1+\frac{4}{\mathsf{OPT}-1}\right)\cdot\max\left\{1+\frac{6}{m},1+\frac{4}{\mathsf{OPT}}\right\}\\
&\leq \rho(G)\cdot\left(1+\frac{12}{\mathsf{OPT}-1}\right).
\end{aligned}$$

Using similar calculations, we can see that

$$\begin{aligned}
\rho(G^{-e}) &\geq \rho(G)\cdot\left(1-\frac{3}{m}\right)\cdot\min\left\{1-\frac{4}{\mathsf{OPT}-1},1-\frac{1}{m-1}\right\}\\
&\geq \rho(G)\cdot\left(1-\frac{7}{\mathsf{OPT}-1}\right).
\end{aligned}$$

Thus, for all $e\in E$, we have that $|\rho(G)-\rho(G^{-e})|\leq\max\left\{\frac{7}{\mathsf{OPT}-1},\frac{12}{\mathsf{OPT}-1}\right\}\cdot\rho(G)=\frac{12\rho(G)}{\mathsf{OPT}-1}$. Hence, $\mathbb{E}_{e\sim E}[|\rho(G)-\rho(G^{-e})|]\leq\frac{12\rho(G)}{\mathsf{OPT}-1}$.

Therefore, the average sensitivity of Algorithm 7.4 is at most

$$\begin{aligned}
&\frac{O(f(G))\cdot g(G)+O(g(G))\cdot f(G)}{f(G)+g(G)}+2\mathsf{OPT}\cdot\mathbb{E}_{e\sim E}[|\rho(G)-\rho(G^{-e})|]\\
&=O\left(\frac{f(G)^{3/4}g(G)^{1/4}}{\frac{g(G)^{1/4}}{f(G)^{1/4}}+\frac{f(G)^{3/4}}{g(G)^{3/4}}}\right)+O\left(\frac{\mathsf{OPT}\rho(G)}{\mathsf{OPT}}\right)\\
&=O\left(f(G)^{3/4}g(G)^{1/4}\right)+O(1)\\
&=O\left(\left(\frac{\mathsf{OPT}^2}{m}\right)^{3/4}\cdot\left((\varepsilon\log n)^{1/4}+\left(\frac{m^3}{\varepsilon^3\mathsf{OPT}^3}\right)^{1/4}\right)\right)\\
&=O\left(\left(\frac{\mathsf{OPT}^{3/2}}{m^{3/4}}\varepsilon^{1/4}\log^{1/4}n+\frac{\mathsf{OPT}^{3/2}}{m^{3/4}}\frac{m^{3/4}}{\varepsilon^{3/4}\mathsf{OPT}^{3/4}}\right)\right)\\
&=O\left(\frac{\mathsf{OPT}^{3/2}}{m^{3/4}}\varepsilon^{1/4}\log^{1/4}n+\frac{\mathsf{OPT}^{3/4}}{\varepsilon^{3/4}}\right)\\
&=O\left(\mathsf{OPT}^{3/4}\left(\varepsilon^{1/4}\log^{1/4}n+\frac{1}{\varepsilon^{3/4}}\right)\right).
\end{aligned}$$

□

7.8.3 Matching Algorithm based on Augmenting Paths

In this section, we describe a $(1-\varepsilon)$-approximation algorithm for the maximum matching problem with average sensitivity $\tilde{O}\left(\mathsf{OPT}^{\frac{c}{c+1}}/\varepsilon^{\frac{3c}{c+1}}\right)$ for $c = O(1/\varepsilon^2)$ in Theorem 7.8.14. The basic building block is a $(1-\varepsilon)$-approximation algorithm for maximum matching that is based on iteratively augmenting a matching with greedily chosen augmenting paths of increasing lengths. In Theorem 7.8.12, we show that the average sensitivity of this algorithm is $\Delta^{O(1/\varepsilon^2)}$, where Δ is the maximum degree of the input graph. To this, we first apply Theorem 7.8.4 to obtain Theorem 7.8.13. We then combine the algorithm guaranteed by Theorem 7.8.13 with the algorithm guaranteed by Theorem 7.8.1 to obtain Theorem 7.8.14.

Algorithm 7.5 Greedy Augmenting Paths Algorithm

```
Input: undirected unweighted graph $G = (V, E)$, parameter $\varepsilon \in (0, 1)$
1: $M_0 \leftarrow \emptyset$.
2: for $i \in \left[\lceil \frac{1}{\varepsilon} - 1 \rceil\right]$ do
3:     Let $A_i$ denote the set of augmenting paths of length $2i-1$ for the matching
       $M_{i-1}$.
4:     Let $A'_i$ denote a maximal set of disjoint paths from $A_i$, where $A'_i$ is made from
       a random ordering of $A_i$.
5:     $M_i \leftarrow M_{i-1} \triangle A'_i$.
6: return $M_{\lceil \frac{1}{\varepsilon} - 1 \rceil}$.
```

Theorem 7.8.12. *Algorithm 7.5 with parameter $\varepsilon > 0$ has approximation ratio $1-\varepsilon$ and average sensitivity $\Delta^{O(1/\varepsilon^2)}$, where Δ is the maximum degree of the input graph.*

Proof. It is known (Garey & Johnson (1979)) that for all $k \geq 0$, $|M_k| \geq \frac{k}{k+1} \cdot |M^*|$, where M^* denotes a maximum matching in G. Hence, the matching $M_{\lceil \frac{1}{\varepsilon} - 1 \rceil}$ is a $(1-\varepsilon)$-approximation to M^*.

(Yoshida et al., 2012, Theorem 3.7) show that for all $k \geq 0$, determining whether a uniformly random edge $e \sim E$ belongs to M_k can be done by querying at most $\Delta^{O(k^2)}$

edges in expectation, where Δ is the maximum degree of G. Applying Theorem 7.2.5 to this, we can see that the average sensitivity of Algorithm 7.5 with parameter $\varepsilon > 0$ and input G is $\Delta^{O(1/\varepsilon^2)}$, where Δ is the maximum degree of G. □

Theorem 7.8.13. *Let* $\varepsilon \in (0,1)$ *be a parameter. There exists an algorithm with approximation ratio* $1-\varepsilon$ *and average sensitivity*

$$O\left(\frac{\varepsilon}{1-\varepsilon}\log n\right) + \left(\frac{m}{\varepsilon^3 \mathsf{OPT}}\right)^{O(1/\varepsilon^2)}.$$

Proof. The algorithm guaranteed by the theorem statement is as follows.

Algorithm $\mathcal{A}_\varepsilon$: On input $G = (V, E)$,

1. Compute OPT.
2. If $\mathsf{OPT} \le \frac{2}{\varepsilon} + 1$ or $m \le \frac{1}{3\varepsilon}$, then output an arbitrary maximum matching.
3. Otherwise, run the algorithm obtained by applying Theorem 7.8.4 with the setting $\tau := \tau(G) = \frac{m}{\varepsilon' \mathsf{OPT}}$ and $\delta := \frac{1}{2\ln n}$ to Algorithm 7.5 run with parameter ε', where $\varepsilon' = \frac{\varepsilon}{3} - \frac{1}{3\mathsf{OPT}}$.

As the analysis is almost identical to that of Theorem 7.8.8, we only highlight the differences.

Approximation guarantee: The analysis of the approximation ratio is straightforward.

Average sensitivity: We focus on the case that $\mathsf{OPT} > \frac{2}{\varepsilon} + 1$ and $m > \frac{1}{3\varepsilon}$ since the average sensitivity in other case is straightforward to analyze. The first term of (7.6) can be bounded by $O\left(\frac{\varepsilon}{1-\varepsilon} \cdot \log n\right)$ in the same way as in the proof of Theorem 7.8.8.

Let $c = O(1/\varepsilon^2)$. Then, the second term of (7.6) becomes

$$\int_0^\infty (2x-2)^2 x^c \frac{1}{2\delta\tau} \exp\left(-\frac{|x-\tau|}{\delta\tau}\right) \mathrm{d}x = 4\int_\tau^\infty x^{c+2} \frac{1}{\delta\tau} \exp\left(-\frac{x-\tau}{\delta\tau}\right) \mathrm{d}x$$

$$= \exp\left(\frac{1}{\delta}\right) (\delta\tau)^{c+2} \Gamma\left(c+3, \frac{1}{\delta}\right) = (\delta\tau)^{c+2}(c+2)! \sum_{k=0}^{c+2} \frac{(1/\delta)^k}{k!} = \left(\frac{m}{\varepsilon^3 \mathsf{OPT}}\right)^{O(1/\varepsilon^2)}$$

where $\Gamma(\cdot,\cdot)$ is the incomplete Gamma function and we have used the fact that $\Gamma(s+1,x) = s!\exp(-x)\sum_{k=0}^{s} x^k/k!$ if s is a non-negative integer. Moreover, each term in the summation $\delta^{c+2}\cdot(c+2)!\sum_{k=0}^{c+2}\frac{(1/\delta)^k}{k!}$ is $o(1)$. Hence, the summation is $O(\frac{1}{\varepsilon^2})$. □

By combining Theorems 7.8.1 and 7.8.13, we get the following.

Theorem 7.8.14. *Let $\varepsilon \in (0,1)$ be a parameter. There exists an algorithm with approximation ratio $1-\varepsilon$ and average sensitivity*

$$\mathrm{OPT}(G)^{\frac{c}{c+1}} \cdot O\left(\left(\frac{\varepsilon}{1-\varepsilon}\cdot \log n\right)^{\frac{1}{c+1}} + \frac{1}{\varepsilon^{\frac{3c}{c+1}}}\right)$$

for $c = O(1/\varepsilon^2)$.

Proof. Let $f(G) = \frac{\mathsf{OPT}^2}{m}$ and $g(G) = \frac{\varepsilon}{1-\varepsilon}\cdot\log n + \left(\frac{m}{\varepsilon^3\mathsf{OPT}}\right)^c$ for $c = O(1/\varepsilon^2)$. Let $\rho(G)$ denote $\frac{g(G)}{f(G)+g(G)}$. Given a graph $G=(V,E)$ as input and a parameter $\varepsilon \in (0,1)$, the algorithm guaranteed by the theorem first computes OPT and returns an arbitrary maximum matching if $\mathsf{OPT} < 2c$ or $m < 2c$. Otherwise, it runs the algorithm given by Theorem 7.8.1 with probability $\rho(G)$ and the algorithm given by Theorem 7.8.13 using the parameter ε with probability $1-\rho(G)$.

We highlight the differences in the analysis when compared to the proof of Theorem 7.8.9, which arise only in the part where we analyze the average sensitivity for the case that $\mathsf{OPT} \geq 2c$ and $m \geq 2c$. While bounding the second term of (7.7), the bounds on $f(G^{-e})$ that we use are identical to that in Claim 7.8.11. The following claim gives bounds for $g(G^{-e})$.

Claim 7.8.15. *For every graph $G = (V, E)$ such that $\mathsf{OPT} \geq c + 1$ and $m \geq 2c$, and for every $e \in E$,*

$$\left(1 - \frac{c}{m}\right) \cdot g(G) \leq g(G^{-e}) \leq \left(1 + \frac{c}{\mathsf{OPT} - c}\right) \cdot g(G).$$

The proof of Claim 7.8.15 is nearly identical to that of Claim 7.8.10. In order to get the upper bound, we use the fact that $(1 + x)^r \leq 1 + \frac{rx}{1-(r-1)x}$ for $x \in [0, \frac{1}{r-1})$ and $r > 1$.

Using these bounds, we can argue that $\mathbb{E}_{e\sim E}[|\rho(G) - \rho(G^{-e})|] \leq \frac{3c}{\mathsf{OPT}-c}$. Therefore, the second term of (7.7) can be bounded by $\frac{3c\mathsf{OPT}}{\mathsf{OPT}-c}$. This is $O(1/\varepsilon^2)$, since $\frac{\mathsf{OPT}}{\mathsf{OPT}-c} \leq 2$ as $\mathsf{OPT} \geq 2c$.

To bound the first term of (7.7), we divide both the numerator and denominator by $f(G)^{\frac{1}{c+1}} \cdot g(G)^{\frac{c}{c+1}}$ and upper bound the resulting fraction by its numerator $g(G)^{\frac{1}{c+1}} \cdot f(G)^{\frac{c}{c+1}}$, which can be simplified to obtain the final upper bound on the average sensitivity. $\square$

7.8.4 Lower bound

In this section, we show a lower bound of $\Omega(n)$ for the problem of exactly computing the maximum matching in a graph.

Theorem 7.8.16. *Every algorithm that exactly computes the maximum matching in a graph has average sensitivity $\Omega(n)$.*

Proof. Let $n \in \mathbb{N}$ be even. Consider the cycle C_n on n vertices. C_n has exactly two maximum matchings M_1 and M_2 of size $n/2$ each. Both M_1 and M_2 consist of alternating edges of the cycle. Let A be an algorithm that outputs M_1 with probability

p and M_2 with probability $1-p$. Assume, without loss of generality, that $p \geq \frac{1}{2}$. For every edge $e \in M_1$, the unique maximum matching in the odd-length path $G-e$ has Hamming distance $n-1$ from M_1. Thus, for each $e \in M_1$, the earth mover's distance between $A(G)$ and $A(G-e)$ is at least $\frac{n-1}{2}$. Hence, the average sensitivity of A is at least $\frac{1}{n}\sum_{e \in M_1} \frac{n-1}{2} = \Omega(n)$. □

7.9 MINIMUM VERTEX COVER

A vertex set $S \subseteq V$ in a graph $G=(V,E)$ is called a *vertex cover* if every edge in E is incident to a vertex in S. In the *minimum vertex cover problem*, given a graph G, we want to compute a vertex cover of the minimum size. In this section, we discuss stable-on-average approximation algorithms for the minimum vertex cover problem.

In Section 7.9.1, we give an algorithm that reduces to the maximum matching problem and prove Theorem 7.9.1. Then, in Section 7.9.2, we show another algorithm based on a differentially private algorithm due to Gupta et al. (2010), which has a worse approximation ratio but could have a smaller average sensitivity compared to the first algorithm. In particular, we prove Theorem 7.9.2.

7.9.1 Reduction to the Maximum Matching Problem

It is well known that, for any maximal matching M, the vertex set consisting of all endpoints of edges in M is a 2-approximate vertex cover. Based on this fact, we slightly modify Algorithm 7.4 to obtain a stable-on-average algorithm for the minimum vertex cover problem. Specifically, we show the following.

Theorem 7.9.1. *Let $\varepsilon \in (0,1)$. There exists a $(2+\varepsilon)$-approximation algorithm for* *the minimum vertex cover problem with average sensitivity*

$$O\left(\mathsf{OPT}^{3/4}\left(\varepsilon^{1/4}\log^{1/4} n+\frac{1}{\varepsilon^{3/4}}\right)\right).$$

In particular when $\varepsilon = 1/\log^{1/4} n$, the average sensitivity is $O(\mathsf{OPT}^{3/4}\log^{3/16} n)$.

Proof. Given a graph $G=(V,E)$, let MM denote the size of a maximum matching in G. Let $\mathcal{A}$ denote the algorithm given by Theorem 7.8.9. Our algorithm to approximate vertex cover is a slight modification of $\mathcal{A}$.

Recall that, on input $G=(V,E)$ and parameter $\varepsilon \in (0,1/2)$, algorithm $\mathcal{A}$ runs one of the following algorithms. The first one, denoted by $\mathcal{A}_1$, simply outputs a maximum matching in G. The second one, denoted by $\mathcal{A}_2$, does the following. If $\mathsf{MM} \le \frac{1}{\varepsilon}+1$ or $m \le \frac{1}{2\varepsilon}$, it outputs an arbitrary maximum matching. Otherwise, $\mathcal{A}_2$ constructs a graph $[G]_L$ by removing vertices of degrees at least a threshold $L \sim \mathsf{Lap}(\tau, \delta\tau)$ and then applies the randomized greedy algorithm on $[G]_L$, where $\tau = \frac{2m}{2\varepsilon\mathsf{MM}-1}$, and $\delta = \frac{1}{2\ln n}$.

Our modification to $\mathcal{A}$ is as follows. When we run $\mathcal{A}_1$, we output the vertex set S consisting of the endpoints of the output matching. When we run $\mathcal{A}_2$, we output the set $T = T_1 \cup T_2$, where T_1 is the set of endpoints of the matching and T_2 is the set of vertices removed by $\mathcal{A}_2$.

Approximation guarantee: Clearly, S is a 2-approximate vertex cover. As T_1 is a vertex cover of $[G]_L$, the set T is a vertex cover of G. We now bound the expected size of T. Let OPT and $\mathsf{OPT}([G]_L)$ be the sizes of minimum vertex covers in G and $[G]_L$, respectively. Then, observing that $\mathsf{OPT} \ge \mathsf{OPT}([G]_L) \ge |T_1|/2$, we have $\mathbb{E}[|T_1|] \le 2\mathsf{OPT}$. We now bound $\mathbb{E}[|T_2|]$. By Proposition 7.8.5, we know that the value of $L < (1-\frac{\ln 2}{2})\cdot\tau$ with probability at most $\frac{1}{n}$. Therefore, with probability at least $1-\frac{1}{n}$, the number of vertices removed is at most $\frac{m}{(1-\frac{\ln 2}{2})\cdot\tau}$ and with probability at most $\frac{1}{n}$, the number of vertices removed is at most n. Therefore, the expected cardinality of S_2 is at most $\frac{m}{(2-\ln 2)\cdot\frac{m}{2\varepsilon\mathsf{MM}-1}}\cdot(1-\frac{1}{n})+n\cdot\frac{1}{n}$, which is at most $2\varepsilon\mathsf{MM}$. Note that $\frac{\mathsf{OPT}}{2}\leq\mathsf{MM}\leq\mathsf{OPT}$ as the set of endpoints in a maximum matching is a 2-approximate vertex cover. Therefore, we can see that $\mathbb{E}[|T_2|]\leq 2\varepsilon\mathsf{OPT}$. It follows that

$$\mathbb{E}[|T|]=\mathbb{E}[|T_1|]+\mathbb{E}[|T_2|]\leq 2\mathsf{OPT}+2\varepsilon\cdot\mathsf{OPT}=(2+2\varepsilon)\cdot\mathsf{OPT}.$$

Average sensitivity: The average sensitivity of $\mathcal{A}_1$ even after the modification is $O\left(\frac{\mathsf{OPT}^2}{m}\right)$, since outputting the endpoints of a matching instead of the matching itself can only affect the average sensitivity by a factor of at most 2.

We now bound the average sensitivity of $\mathcal{A}_2$ after the modification. Consider a graph $G=(V,E)$. Let A_2^i for $i\in[2]$ denote the algorithm that simulates the actions of $\mathcal{A}_2$ and outputs only T_i. Let $H_{e,\pi}$ for $e\in E$ denote the Hamming distance between the outputs of $\mathcal{A}_2$ on G and G^{-e} when run using the same random string $\pi\in\{0,1\}^*$. Let $H_{e,\pi}^i$ for $i\in[2]$ and $e\in E$ denote the Hamming distance between the outputs of $\mathcal{A}_2^{(i)}$ on G and G^{-e} when run using the same random string π. Since T_1 and T_2 are always disjoint, we have for all $e\in E$ and all $\pi\in\{0,1\}^*$,

$$H_{e,\pi}\leq H_{e,\pi}^1+H_{e,\pi}^2.$$

Therefore, the average sensitivity of $\mathcal{A}_2$ is at most the sum of average sensitivities of $\mathcal{A}_2^{(1)}$ and $\mathcal{A}_2^{(2)}$.

The average sensitivity of $\mathcal{A}_2^{(1)}$ is $O\left(\frac{\varepsilon}{1-\varepsilon}\cdot\log n+\frac{m^3}{\varepsilon^3\mathsf{MM}^3}\right)$, since outputting the endpoints of a matching does not change the average sensitivity asymptotically.

We now bound the average sensitivity of $\mathcal{A}_2^{(2)}$. Note that the output of $\mathcal{A}_2^{(2)}$ on a graph $G=(V,E)$ is fully characterized by the value of the Laplace random variable that is sampled. Fix $e\in E$. We bound the earth mover's distance between $\mathcal{A}_2^{(2)}(G)$

and $\mathcal{A}_2^{(2)}(G^{-e})$.

Let $f_G(x)$ and $f_{G^{-e}}(x)$ denote the probability densities that $\mathcal{A}_2^{(2)}$ samples the value x as the threshold on inputs G and G^{-e}, respectively. We start with the distribution $\mathcal{A}_2^{(2)}(G)$. For each $x \in \mathbb{R}$, we retain a probability density of $\min\{f_{G^{-e}}(x), f_G(x)\}$ at point x at a cost of $2 \cdot \min\{f_{G^{-e}}(x), f_G(x)\}$. The factor 2 comes from the fact that for the same value of random threshold, the sets output by $\mathcal{A}_2^{(2)}$ on G and G^{-e} can differ in at most 2 vertices.

We also transport a probability density of $\max\{f_G(x) - f_{G^{-e}}(x), 0\}$ to another point where there is a deficit in probability density. The cost of this transport equals to the transported probability density weighted by Hamming distance between the cardinality of outputs at the source and destination points. Let MM_e denote the size of a maximum matching in G^{-e}. As we argued earlier, for the distribution $\mathcal{A}_2^{(2)}(G)$, a probability of at least $1 - \frac{1}{n}$ is located on values of x that would make the cardinality of output of $\mathcal{A}_2^{(2)}$ at most $2\varepsilon\mathsf{MM}$. Using the same argument, we can see that, for the distribution $\mathcal{A}_2^{(2)}(G^{-e})$, a probability of at least $1 - \frac{1}{n}$ is located on values of x that would make the cardinality of output of $\mathcal{A}_2^{(2)}$ at most $2\varepsilon\mathsf{MM}_e \leq 2\varepsilon\mathsf{MM}$. If a probability p is transported from a source point with output size at most $2\varepsilon\mathsf{MM}$ to a destination point with output size at most $2\varepsilon\mathsf{MM}$, the cost of the transport is at most $p \cdot 4\varepsilon\mathsf{MM}$. Only for a probability of at most $\frac{2}{n}$, the symmetric difference between the source and destination output sets is $\Omega(n)$ and hence, the cost of transport is at most 2. Thus, the earth mover's distance between $\mathcal{A}_2^{(2)}(G)$ and $\mathcal{A}_2^{(2)}(G^{-e})$ is at most $d_{\mathrm{TV}}(L, L_e) \cdot 4\varepsilon\mathsf{MM} + O(1)$.

Using Claim 7.8.7 and following the steps in the proof of Theorem 7.8.8, the average sensitivity of $\mathcal{A}_2^{(2)}$ is bounded by $O(\frac{\varepsilon^2}{1-\varepsilon}\log n)$.

Hence, the average sensitivity of $\mathcal{A}_2$ is $O\left(\frac{\varepsilon}{1-\varepsilon} \cdot \log n + \frac{m^3}{\varepsilon^3\mathsf{MM}^3}\right)$. Following the proof of Theorem 7.8.9 again, the overall average sensitivity is bounded by

$$O\left(\mathsf{MM}^{3/4}\left(\varepsilon^{1/4}\log^{1/4}n+\frac{1}{\varepsilon^{3/4}}\right)\right)=O\left(\mathsf{OPT}^{3/4}\left(\varepsilon^{1/4}\log^{1/4}n+\frac{1}{\varepsilon^{3/4}}\right)\right).$$

By replacing ε with $\varepsilon/2$ throughout, we get the approximation and average sensitivity guarantees given by the statement. □

7.9.2 Algorithm based on a Differentially Private Algorithm

We consider another algorithm for the minimum vertex cover problem based on the differentially private algorithm due to Gupta et al. (2010) and show the following.

Theorem 7.9.2. *Let $\varepsilon \geq 0$. There exists an algorithm for the minimum vertex cover problem with average sensitivity $O(n^2/m^{1+\varepsilon})$ and approximation ratio $O(\frac{m^{1+\varepsilon}\log n}{n})$.*

The algorithm of Gupta et al. (2010) is based on the simple (non-private) 2-approximation algorithm due to Pitt (1985) that repeatedly selects a vertex at random with probability proportional to its degree with respect to the uncovered edges and adds the sampled vertex to the solution. To make this algorithm differentially private, Gupta et al. (2010) mixed the distribution with a uniform distribution, using a weight that grows as the number of remaining vertices decreases. Our algorithm, shown in Algorithm 7.6, is similar to theirs though our choice of weights is different.

We will show that Algorithm 7.6 has approximation ratio $O((m^{1+\varepsilon}\log n)/n)$ and average sensitivity $O(n^2/m^{1+\varepsilon})$. Although the approximation ratio is worse than that discussed in Section 7.9.1, the present one has a better sensitivity. In what follows, for simplicity, we assume $n \leq m^{1+\varepsilon}$.

Lemma 7.9.3. *The approximation ratio of Algorithm 7.6 is $O((m^{1+\varepsilon}\log n)/n)$.*

Algorithm 7.6 AVERAGELY STABLE ALGORITHM FOR VERTEX COVER

```
Input: Graph $G = (V, E)$ and parameter $\varepsilon \geq 0$
1: Let $G_1 = (V_1, E_1) \leftarrow G$, and $S \leftarrow \emptyset$.
2: for $i = 1, \ldots, n$ do
3:     $w_i \leftarrow 2m^{1+\varepsilon}/(n - i + 1)$
4:     Sample a vertex $v \in V(G_i)$ with probability proportional to $d_{G_i}(v) + w_i$.
5:     if $d_{G_i}(v) \geq 1$ then
6:         $S \leftarrow S \cup \{v\}$.
7:     $G_{i+1} \leftarrow$ the graph obtained from $G_i$ by removing $v$ and incident edges.
8: return $S$.
```

Proof. It is shown by Gupta et al. (2010) that the approximation ratio is $(2 + (2/n)\sum_{i=1}^{n} w_i)$, which is

$$2 + \frac{2}{n}\sum_{i=1}^{n}\frac{2m^{1+\varepsilon}}{n-i+1} = O\left(\frac{m^{1+\varepsilon}\log n}{n}\right). \qquad \square$$

Lemma 7.9.4. *The average sensitivity of Algorithm 7.6 is $O(n^2/m^{1+\varepsilon})$.*

Proof. For a while, we fix a graph $G = (V, E)$ and an edge $e \in E$. To analyze the sensitivity of Algorithm 7.6, we focus on the vertex ordering $\pi = (v_1, \ldots, v_n)$, where v_i $(i = 1, \ldots, n)$ is the vertex sampled at the i-th step. Note that the output set is fully determined by π. Let $p_G(\pi)$ and $p_{G^{-e}}(\pi)$ denote the probabilities that Algorithm 7.6 samples the vertex ordering π on inputs G and G^{-e}, respectively. Then, we can bound the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$, where $\mathcal{A}$ is Algorithm 7.6, as follows. For each vertex ordering π, we retain a mass of $p_G(\pi)$ at a cost of $1 \cdot p_G(\pi)$ (the output sets can have a Hamming distance of 1 even if the vertex orderings generated are the same) and bring in a "remaining" mass of $\max\{0, p_{G^{-e}}(\pi) - p_G(\pi)\}$ from other points where there is excess probability mass. The second step costs at most $n \cdot \max\{0, p_{G^{-e}}(\pi) - p_G(\pi)\}$.

We now bound $p_{G^{-e}}(\pi) - p_G(\pi)$. Let $k_e \in [n]$ be such that the first endpoint of e in π occurs in the k_e-th position in π. As long as e is fixed, we write k instead for simplicity. Let G_i and G'_i be the graphs at the beginning of the i-th step of Algorithm 7.6 on G and G^{-e}, respectively. Let m_i and m'_i be the number of edges in G_i and G'_i, respectively. Note that $m_i = m'_i + 1$ for $i \le k$ and $m_i = m'_i$ for $i > k$. For a vertex $v \in V$, let $d_i(v)$ and $d'_i(v)$ denote the degrees of v in G_i and G'_i, respectively. Note that $d_i(v_i) = d'_i(v_i)$ for $i \neq k$ and $d_k(v_k) = d'_k(v_k) + 1$. Now, we have

$$
\begin{aligned}
&p_{G^{-e}}(\pi) - p_G(\pi) \\
&= \prod_{i=1}^{n} \frac{d'_i(v_i) + w_i}{(n-i+1)w_i + 2m'_i} - \prod_{i=1}^{n} \frac{d_i(v_i) + w_i}{(n-i+1)w_i + 2m_i} \\
&= \frac{\prod_{i \in [n] : i \neq k} d_i(v_i) + w_i}{\prod_{i=k+1}^{n} (n-i+1)w_i + 2m_i} \left[\frac{d'_k(v_k) + w_k}{\prod_{i=1}^{k} (n-i+1)w_i + 2m'_i} - \frac{d_k(v_k) + w_k}{\prod_{i=1}^{k} (n-i+1)w_i + 2m_i} \right] \\
&= \frac{\prod_{i \in [n] : i \neq k} d_i(v_i) + w_i}{\prod_{i=k+1}^{n} (n-i+1)w_i + 2m_i} \left[\frac{d_k(v_k) - 1 + w_k}{\prod_{i=1}^{k} (n-i+1)w_i + 2m_i - 2} - \frac{d_k(v_k) + w_k}{\prod_{i=1}^{k} (n-i+1)w_i + 2m_i} \right]
\end{aligned}
$$

Let Z_π and Z'_π stand for $\prod_{i \le k} (n-i+1)w_i + 2m_i$ and $\prod_{i \le k} (n-i+1)w_i + 2m_i - 2$, respectively. The expression inside the square brackets can be written as

$$
(d_k(v_k) + w_k) \left(\frac{1}{Z'_\pi} - \frac{1}{Z_\pi} \right) - \frac{1}{Z'_\pi}.
$$

Let t_i stand for $(n-i+1)w_i + 2m_i$ for all $i \le k$. By using the identity

$$
\prod_{i=1}^{n} x_i - \prod_{i=1}^{n} y_i = \sum_{i=1}^{n} (x_i - y_i) \cdot \prod_{j=1}^{i-1} x_j \cdot \prod_{j=i+1}^{n} y_j,
$$

we can rewrite $\frac{1}{Z'_\pi} - \frac{1}{Z_\pi}$ as follows.

$$\begin{aligned}
\frac{1}{Z'_\pi} - \frac{1}{Z_\pi} &= \sum_{i=1}^{k} \left(\prod_{j=1}^{i-1} \frac{1}{t_j - 2}\right) \left(\frac{1}{t_i - 2} - \frac{1}{t_i}\right) \left(\prod_{j=i+1}^{k} \frac{1}{t_j}\right) \\
&= 2\sum_{i=1}^{k} \left(\prod_{j=1}^{i} \frac{1}{t_j - 2}\right) \left(\prod_{j=i}^{k} \frac{1}{t_j}\right) \\
&\le 2\prod_{i=1}^{k} \frac{1}{t_i - 2} \sum_{i=1}^{k} \frac{1}{t_i} = \frac{2}{Z'_\pi} \sum_{i=1}^{k} \frac{1}{t_i} = \frac{2}{Z_\pi} \cdot \frac{Z_\pi}{Z'_\pi} \cdot \sum_{i=1}^{k} \frac{1}{t_i}.
\end{aligned}$$

Note that

$$\begin{aligned}
\frac{Z_\pi}{Z'_\pi} &= \prod_{i=1}^{k} \frac{(n-i+1)w_i + 2m_i}{(n-i+1)w_i + 2m_i - 2} = \prod_{i=1}^{k} \left(1 + \frac{2}{(n-i+1)w_i + 2m_i - 2}\right) \\
&\le \exp\left(\sum_{i=1}^{k} \frac{2}{(n-i+1)w_i + 2m_i - 2}\right) = \exp\left(\sum_{i=1}^{k} \frac{2}{2m^{1+\varepsilon} + 2m_i - 2}\right) \\
&\le \exp\left(\frac{n}{m^{1+\varepsilon}}\right) \le e. \qquad (\because n \le m^{1+\varepsilon})
\end{aligned}$$

Combining all of the above, we get,

$$\begin{aligned}
p_{G^{-e}}(\pi) - p_G(\pi) &= \frac{\prod_{i\in[n]:i\neq k} d_i(v_i) + w_i}{\prod_{i=k+1}^{n} (n-i+1)w_i + 2m_i} \left[(d_k(v_k) + w_k)\left(\frac{1}{Z'_\pi} - \frac{1}{Z_\pi}\right) - \frac{1}{Z'_\pi}\right] \\
&\le \frac{\prod_{i\in[n]:i\neq k} d_i(v_i) + w_i}{\prod_{i=k+1}^{n} (n-i+1)w_i + 2m_i} \left[(d_k(v_k) + w_k)\left(\frac{2}{Z_\pi} \cdot \frac{Z_\pi}{Z'_\pi} \cdot \sum_{i\le k} \frac{1}{t_i}\right)\right] \\
&\le 2ep_G(\pi) \sum_{i\le k} \frac{1}{t_i}.
\end{aligned}$$

Now, we take the average over $e \in E$.

$$\begin{aligned}
\mathbb{E}_e\left[p_{G^{-e}}(\pi) - p_G(\pi)\right] &\le \frac{1}{m} \sum_{e\in E} \left(2ep_G(\pi) \sum_{i=1}^{k_e} \frac{1}{t_i}\right) = \frac{2ep_G(\pi)}{m} \sum_{j=1}^{n} \left(d_j(v_j) \sum_{i=1}^{j} \frac{1}{t_i}\right) \\
&= \frac{2ep_G(\pi)}{m} \sum_{i=1}^{n} \frac{1}{t_i} \sum_{j=i}^{n} d_j(v_j) = \frac{2ep_G(\pi)}{m} \sum_{i=1}^{n} \frac{m_i}{t_i}
\end{aligned}$$

$$= \frac{2ep_G(\pi)}{m}\sum_{i=1}^{n}\frac{m_i}{(n-i+1)w_i+2m_i} \leq \frac{2ep_G(\pi)}{m}\sum_{i=1}^{n}\frac{m_i}{2m^{1+\varepsilon}+2m_i}$$
$$\leq \frac{ep_G(\pi)}{m}\sum_{i=1}^{n}\frac{1}{m^{\varepsilon}} \leq \frac{enp_G(\pi)}{m^{1+\varepsilon}}.$$

Then, the average sensitivity is bounded by

$$1 + n\sum_{\pi}\mathbb{E}_e\left[p_{G^{-e}}(\pi) - p_G(\pi)\right] = 1 + \sum_{\pi}\frac{en^2p_G(\pi)}{m^{1+\varepsilon}} = O\left(\frac{n^2}{m^{1+\varepsilon}}\right). \qquad \square$$

7.10 2-COLORING

In the 2-*coloring problem*, given a bipartite graph $G = (V, E)$, we are to output a (proper) 2-coloring on G, that is, an assignment $f : V \to \{0, 1\}$ such that $f(u) \neq f(v)$ for every edge $(u, v) \in E$. Clearly this problem can be solved in linear time. In this section, however, we show that there is no stable-on-average algorithm for the 2-coloring problem.

Theorem 7.10.1. *Any (randomized) algorithm for the* 2*-coloring problem has average sensitivity* $\Omega(n)$*.*

Proof. Suppose that there is a (randomized) algorithm $\mathcal{A}$ whose average sensitivity is at most βn for $\beta < 1/256$. In what follows, we assume that n, that is, the number of vertices in the input graph, is a multiple of 16.

Let $\mathcal{P}_n$ be the family of all possible paths on n vertices, and let $\mathcal{Q}_n$ be the family of all possible graphs on n vertices consisting of two paths. Note that $|\mathcal{P}_n| = n!/2$ and $|\mathcal{Q}_n| = (n-1)n!/4$. Consider a bipartite graph $H = (\mathcal{P}_n, \mathcal{Q}_n; E)$, where a pair (P, Q) is in E if and only if Q can be obtained by removing an edge in P. Note that each $P \in \mathcal{P}_n$ has $n-1$ neighbors in H and each $Q \in \mathcal{Q}_n$ has four neighbors in H.

We say that an edge $(P, Q) \in E$ is *intimate* if $d_{\mathrm{EM}}\big(\mathcal{A}(P), \mathcal{A}(Q)\big) \leq 8\beta n$. We observe that for every $P \in \mathcal{P}_n$, at least a 7/8-fraction of the edges incident to P are intimate; otherwise

$$\mathbb{E}_{e \in E(P)}\left[d_{\mathrm{EM}}\big(\mathcal{A}(P), \mathcal{A}(P-e)\big)\right] > \frac{1}{8} \cdot 8\beta n = \beta n,$$

which is a contradiction, where $E(P)$ denotes the set of edges in P.

We say that a graph $Q \in \mathcal{Q}_n$ is *heavy* if both components of Q have at least $n/16$ vertices, and say that an edge $(P, Q) \in E$ is *heavy* if Q is heavy. We observe that for every $P \in \mathcal{P}_n$, at least a $7/8$-fraction of the edges incident to P are heavy.

We say that an edge $(P, Q) \in E$ is *good* if it is intimate and heavy. Observe that for every $P \in \mathcal{P}_n$, by the union bound, at least a $3/4$-fraction of the edges incident to P are good. In particular, this means that the fraction of good edges in H is at least $3/4$. Hence, there exists $Q^* \in \mathcal{Q}_n$ that has at least three good incident edges; otherwise the fraction of good edges in H is at most $2/4 = 1/2$, which is a contradiction.

Let $f_1, \ldots, f_4$ be the four 2-colorings of Q^*. As Q^* has three good incident edges, without loss of generality, there are adjacent paths $P_1, P_2 \in \mathcal{P}_n$ such that both (P_1, Q^*) and (P_2, Q^*) are good, and there is no assignment that is a 2-coloring for both P_1 and P_2. Without loss of generality, we assume that f_1, f_2 are 2-colorings of P_1, and f_3, f_4 are 2-colorings of P_2. Note that $d_{\mathrm{Ham}}(f_i, f_j) \geq n/16$ for $i \neq j$ because Q is heavy. Let $q_i = \Pr[\mathcal{A}(Q^*) = f_i]$ for $i \in [4]$. As the edge (P_1, Q^*) is intimate, we have

$$\begin{aligned}
8\beta n &\geq d_{\mathrm{EM}}\big(\mathcal{A}(P_1), \mathcal{A}(Q^*)\big) \\
&\geq \frac{n}{16}\left(\left|\Pr[\mathcal{A}(P_1) = f_1] - q_1\right| + \left|\Pr[\mathcal{A}(P_1) = f_2] - q_2\right| + q_3 + q_4\right) \\
&= \frac{n}{16}\left(\left|\Pr[\mathcal{A}(P_1) = f_1] - q_1\right| + \left|\Pr[\mathcal{A}(P_1) = f_2] - q_2\right| + 1 - q_1 - q_2\right)
\end{aligned}$$

and hence we must have $q_1 + q_2 \geq 1 - 128\beta$. Considering $d_{\mathrm{EM}}\big(\mathcal{A}(P_2), \mathcal{A}(Q^*)\big)$, we also have $q_3 + q_4 \geq 1 - 128\beta$. However,

$$1 = q_1 + q_2 + q_3 + q_4 \geq (1 - 128\beta) + (1 - 128\beta) = 2 - 256\beta > 1$$

as $\beta < 1/256$, which is a contradiction. $\square$

7.11 GENERAL RESULTS ON AVERAGE SENSITIVITY

In this section, we state and prove some basic properties of average sensitivity and show that locality guarantees of solutions output by an algorithm imply low average sensitivity for that algorithm.

7.11.1 k-Average Sensitivity from Average Sensitivity

In this section, we prove Theorem 7.2.1, which says that, if an algorithm is stable-on-average against deleting a single edge, it is also stable-on-average against deleting multiple edges. We restate the theorem here.

Theorem 7.2.1. *Let $\mathcal{A}$ be an algorithm for a graph problem with the average sensitivity given by $f(n, m)$. Then, for any integer $k \geq 1$, the algorithm $\mathcal{A}$ has k-average sensitivity at most $\sum_{i=1}^{k} f(n, m - i + 1)$.*

Proof. We have

$$\begin{aligned}
&\mathbb{E}_{\{e_1,\ldots,e_k\}\sim\binom{E}{k}} \left[d_{\mathrm{EM}}\big(\mathcal{A}(G), \mathcal{A}(G - \{e_1, \ldots, e_k\})\big)\right] \\
\leq &\mathbb{E}_{\{e_1,\ldots,e_k\}\sim\binom{E}{k}} \left[\sum_{i=1}^{k} d_{\mathrm{EM}}\big(\mathcal{A}(G - \{e_1, \ldots, e_{i-1}\}), \mathcal{A}(G - \{e_1, \ldots, e_i\})\big)\right] \\
= &\mathbb{E}_{e_1\sim E}\Big[d_{\mathrm{EM}}\big(\mathcal{A}(G), \mathcal{A}(G - \{e_1\})\big) + \mathbb{E}_{e_2\sim E}\Big[d_{\mathrm{EM}}\Big(\mathcal{A}(G - \{e_1\}), \mathcal{A}(G - \{e_1, e_2\})\big) + \\
&\quad \cdots + \mathbb{E}_{e_k\sim E}\Big[d_{\mathrm{EM}}\big(\mathcal{A}(G - \{e_1, \ldots, e_{k-1}\}), \mathcal{A}(G - \{e_1, \ldots, e_k\})\big) \ldots\Big]\Big]\Big]
\end{aligned}$$

$$
\begin{aligned}
=& f(n,m) + \mathbb{E}_{e_1 \sim E}\Big[\beta(G - \{e_1\}) + \mathbb{E}_{e_2 \sim E}\Big[\beta(G - \{e_1, e_2\}) + \cdots \\
&\qquad + \mathbb{E}_{e_{k-1} \sim E}\Big[\beta(G - \{e_1, \ldots, e_{k-1}\}) \ldots \Big]\Big]\Big] \\
\leq& \sum_{i=1}^{k} f(n, m - i + 1).
\end{aligned}
$$

Here, the first inequality is due to the triangle inequality. □

7.11.2 Sequential Composition

In this section, we state and prove our two sequential composition theorems Theorem 7.2.2 and Theorem 7.2.3.

Theorem 7.2.2 (Sequential composition)**.** *Consider two randomized algorithms $\mathcal{A}_1 : \mathcal{G} \to \mathcal{S}_1, \mathcal{A}_2 : \mathcal{G} \times \mathcal{S}_1 \to \mathcal{S}_2$. Suppose that the average sensitivity of $\mathcal{A}_1$ with respect to the total variation distance is γ_1 and the average sensitivity of $\mathcal{A}_2(\cdot, S_1)$ is $\beta_2^{(S_1)}$ for any $S_1 \in \mathcal{S}_1$. Let $\mathcal{A} : \mathcal{G} \to \mathcal{S}_2$ be a randomized algorithm obtained by composing $\mathcal{A}_1$ and $\mathcal{A}_2$, that is, $\mathcal{A}(G) = \mathcal{A}_2(G, \mathcal{A}_1(G))$. Then, the average sensitivity of $\mathcal{A}$ is $\mathsf{H} \cdot \gamma_1(G) + \mathbb{E}_{S_1 \sim \mathcal{A}_1(G)}\left[\beta_2^{(S_1)}(G)\right]$, where H denotes the maximum Hamming weight among those of solutions obtained by running $\mathcal{A}$ on G and G^{-e} over all $e \in E$.*

Proof. Consider $G = (V, E)$ and let $e \in E$. We bound the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$ as follows. For a distribution $\mathcal{D}$, we use $f_{\mathcal{D}}$ to denote its probability mass function. We know that for all $S_1 \in \mathcal{S}_1$ and $S_2 \in \mathcal{S}_2$

$$
f_{(\mathcal{A}_1(G), \mathcal{A}_2(G, S_1))}(S_1, S_2) = f_{\mathcal{A}_1(G)}(S_1) \cdot f_{\mathcal{A}_2(G, S_1)}(S_2),
$$

where $(\mathcal{A}_1(G), \mathcal{A}_2(G, S_1))$ denotes the joint distribution of $\mathcal{A}_1(G)$ and $\mathcal{A}_2(G, S_1)$. Fix $S_1 \in \mathcal{S}_1$. For $S_2 \in \mathcal{S}_2$, we transform probabilities of the form $f_{(\mathcal{A}_1(G), \mathcal{A}_2(G, S_1))}(S_1, S_2)$ to $f_{\mathcal{A}_1(G)}(S_1) \cdot f_{\mathcal{A}_2(G^{-e}, S_1)}(S_2)$, for a total cost of $f_{\mathcal{A}_1(G)}(S_1) \cdot d_{\mathrm{EM}}(\mathcal{A}_2(G, S_1), \mathcal{A}_2(G^{-e}, S_1))$.

We can now, for each $S_1 \in \mathcal{S}_1$ and $S_2 \in \mathcal{S}_2$, transform the probability $f_{\mathcal{A}_1(G)}(S_1) \cdot f_{\mathcal{A}_2(G^{-e},S_1)}(S_2)$ into $f_{\mathcal{A}_1(G^{-e})}(S_1) \cdot f_{\mathcal{A}_2(G^{-e},S_1)}(S_2)$ at cost at most $d_{\mathrm{TV}}(\mathcal{A}_1(G), \mathcal{A}_1(G^{-e})) \cdot \mathsf{H}$, where H denotes the maximum Hamming weight among those of solutions obtained by running $\mathcal{A}$ on G and $\{G^{-e}\}_{e \in E}$. Thus, the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$ is at most

$$d_{\mathrm{TV}}(\mathcal{A}_1(G), \mathcal{A}_1(G^{-e})) \cdot \mathsf{H} + \int\limits_{\mathcal{S}_1} f_{\mathcal{A}_1(G)}(S_1) \cdot d_{\mathrm{EM}}\big(\mathcal{A}_2(G, S_1), \mathcal{A}_2(G^{-e}, S_1)\big)\ \mathrm{d}S_1.$$

Hence, the average sensitivity of $\mathcal{A}$ can be bounded as:

$$\begin{aligned}
&\mathbb{E}_{e \sim E}\left[d_{\mathrm{EM}}(\mathcal{A}(G), \mathcal{A}(G^{-e}))\right] \\
\leq&\mathsf{H} \cdot \mathbb{E}_{e \sim E}\left[d_{\mathrm{TV}}(\mathcal{A}_1(G), \mathcal{A}_1(G^{-e}))\right] \\
&\qquad + \mathbb{E}_{e \sim E}\left[\int\limits_{S_1 \in \mathcal{S}_1} f_{\mathcal{A}_1(G)}(S_1) \cdot d_{\mathrm{EM}}(\mathcal{A}_2(G, S_1), \mathcal{A}_2(G^{-e}, S_1))\ \mathrm{d}S_1\right] \\
\leq&\mathsf{H}\gamma_1(G) + \mathbb{E}_{S_1 \sim \mathcal{A}_1(G)}\left[d_{\mathrm{EM}}(\mathcal{A}_2(G, S_1), \mathcal{A}_2(G^{-e}, S_1))\right] \\
=&\mathsf{H}\gamma_1(G) + \mathbb{E}_{S_1 \sim \mathcal{A}_1(G)}\left[\mathbb{E}_{e \sim E} d_{\mathrm{EM}}(\mathcal{A}_2(G, S_1), \mathcal{A}_2(G^{-e}, S_1))\right] \\
=&\mathsf{H}\gamma_1(G) + \mathbb{E}_{S_1 \sim \mathcal{A}_1(G)}\left[\beta_2^{(S_1)}(G)\right].
\end{aligned}$$

We are able to interchange the order of expectations because of Fubini's theorem (Fubini (1907)). □

The following theorem states the composition of average sensitivity with respect to the total variation distance.

Theorem 7.2.3 (Sequential composition w.r.t. the TV distance)**.** *Consider k randomized algorithms $\mathcal{A}_i : \mathcal{G} \times \prod_{j=1}^{i-1} \mathcal{S}_j \to \mathcal{S}_i$ for $i \in [k]$. Suppose that, for each $i \in [k]$, the average sensitivity of $\mathcal{A}_i(\cdot, S_1, \ldots, S_{i-1})$ is γ_i with respect to the total variation* *distance for every $S_1 \in \mathcal{S}_1, \dots, S_{i-1} \in \mathcal{S}_{i-1}$. Consider a sequence of computations $S_1 = \mathcal{A}_1(G), S_2 = \mathcal{A}_2(G, S_1), \dots, S_k = \mathcal{A}_k(G, S_1, \dots, S_{k-1})$. Let $\mathcal{A} : \mathcal{G} \to \mathcal{S}_k$ be a randomized algorithm that performs this sequence of computations on input G and outputs S_k. Then, the average sensitivity of $\mathcal{A}$ with respect to the total variation distance is at most $\sum_{i=1}^{k} \gamma_i(G)$.*

Theorem 7.2.3 can be immediately obtained by iteratively applying Lemma 7.11.1.

Lemma 7.11.1. *Consider two randomized algorithms $\mathcal{A}_1 : \mathcal{G} \to \mathcal{S}_1, \mathcal{A}_2 : \mathcal{G} \times \mathcal{S}_1 \to \mathcal{S}_2$ for a graph problem. Suppose that the average sensitivity of $\mathcal{A}_1$ is $\gamma_1(G)$ and the average sensitivity of $\mathcal{A}_2(\cdot, S_1)$ is $\gamma_2(G)$ for any $S_1 \in \mathcal{S}_1$, both with respect to the total variation distance. Let $\mathcal{A} : \mathcal{G} \to \mathcal{S}_2$ be a randomized algorithm obtained by composing $\mathcal{A}_1$ and $\mathcal{A}_2$, that is, $\mathcal{A}(G) = \mathcal{A}_2(G, \mathcal{A}_1(G))$. Then, the average sensitivity of $\mathcal{A}$ is $\gamma_1(G) + \gamma_2(G)$ with respect to the total variation distance.*

Proof. For a distribution $\mathcal{D}$, we use $f_{\mathcal{D}}$ to denote its probability mass function. Consider a graph $G = (V, E)$. Note that

$$f_{\mathcal{A}(G)}(S_2) = \int_{\mathcal{S}_1} f_{\mathcal{A}_2(G,S_1)}(S_2) f_{\mathcal{A}_1(G)}(S_1) \, \mathrm{d}S_1.$$

Then we have that, for $e \in E$,

$$\begin{aligned}
& d_{\mathrm{TV}}\big(\mathcal{A}(G), \mathcal{A}(G^{-e})\big) \\
& = \frac{1}{2} \int_{\mathcal{S}_2} \left| \int_{\mathcal{S}_1} f_{\mathcal{A}_2(G,S_1)}(S_2) f_{\mathcal{A}_1(G)}(S_1) \, \mathrm{d}S_1 - \int_{\mathcal{S}_1} f_{\mathcal{A}_2(G^{-e},S_1)}(S_2) f_{\mathcal{A}_1(G^{-e})}(S_1) \, \mathrm{d}S_1 \right| \mathrm{d}S_2 \\
& = \frac{1}{2} \int_{\mathcal{S}_2} \Bigg| \int_{\mathcal{S}_1} f_{\mathcal{A}_2(G,S_1)}(S_2) \Big(f_{\mathcal{A}_1(G)}(S_1) - f_{\mathcal{A}_1(G^{-e})}(S_1) \Big) \, \mathrm{d}S_1 - \\
& \qquad \int_{\mathcal{S}_1} \Big(f_{\mathcal{A}_2(G^{-e},S_1)}(S_2) - f_{\mathcal{A}_2(G,S_1)}(S_2) \Big) f_{\mathcal{A}_1(G^{-e})}(S_1) \, \mathrm{d}S_1 \Bigg| \, \mathrm{d}S_2
\end{aligned}$$

$$
\begin{aligned}
&\leq \frac{1}{2}\int\limits_{\mathcal{S}_1}\left|f_{\mathcal{A}_1(G)}(S_1) - f_{\mathcal{A}_1(G^{-e})}(S_1)\right| \mathrm{d}S_1 \cdot \int\limits_{\mathcal{S}_2} f_{\mathcal{A}_2(G,S_1)}(S_2)\,\mathrm{d}S_2 + \\
&\quad \int\limits_{\mathcal{S}_1} f_{\mathcal{A}_1(G^{-e})}(S_1)\,\mathrm{d}S_1 \cdot \frac{1}{2}\int\limits_{\mathcal{S}_2}\left|f_{\mathcal{A}_2(G^{-e},S_1)}(S_2) - f_{\mathcal{A}_2(G,S_1)}(S_2)\right| \mathrm{d}S_2 \\
&= \frac{1}{2}\int\limits_{\mathcal{S}_1}\left|f_{\mathcal{A}_1(G)}(S_1) - f_{\mathcal{A}_1(G^{-e})}(S_1)\right| \mathrm{d}S_1 + \\
&\quad \int\limits_{\mathcal{S}_1} f_{\mathcal{A}_1(G^{-e})}(S_1)\,\mathrm{d}S_1 \cdot \frac{1}{2}\int\limits_{\mathcal{S}_2}\left|f_{\mathcal{A}_2(G^{-e},S_1)}(S_2) - f_{\mathcal{A}_2(G,S_1)}(S_2)\right| \mathrm{d}S_2 \\
&= d_{\mathrm{TV}}\big(\mathcal{A}_1(G), \mathcal{A}_1(G^{-e})\big) + \int\limits_{\mathcal{S}_1} f_{\mathcal{A}_1(G^{-e})}(S_1)\cdot d_{\mathrm{TV}}\big(\mathcal{A}_2(G,S_1), \mathcal{A}_2(G^{-e},S_1)\big)\,\mathrm{d}S_1.
\end{aligned}
$$

Hence, the average sensitivity of $\mathcal{A}$ with respect to the total variation distance can be bounded as,

$$
\begin{aligned}
&\mathbb{E}_{e\sim E}\left[d_{\mathrm{TV}}\big(\mathcal{A}(G), \mathcal{A}(G^{-e})\big)\right] \\
&\leq \mathbb{E}_{e\sim E}\left[d_{\mathrm{TV}}\big(\mathcal{A}_1(G), \mathcal{A}_1(G^{-e})\big)\right] \\
&\qquad + \mathbb{E}_{e\sim E}\left[\int\limits_{\mathcal{S}_1} f_{\mathcal{A}_1(G^{-e})}(S_1)\cdot d_{\mathrm{TV}}\big(\mathcal{A}_2(G,S_1), \mathcal{A}_2(G^{-e},S_1)\big)\,\mathrm{d}S_1\right] \\
&\leq \gamma_1(G) + \int\limits_{\mathcal{S}_1} f_{\mathcal{A}_1(G^{-e})}(S_1)\,\mathrm{d}S_1 \cdot \gamma_2(G) \\
&= \gamma_1(G) + \gamma_2(G).
\end{aligned}
$$

□

7.11.3 Parallel Composition

In this section, we prove Theorem 7.2.4, which bounds the average sensitivity of an algorithm obtained by running different algorithms according to a distribution in terms of the average sensitivities of the component algorithms. We restate the theorem here.

Theorem 7.2.4 (Parallel composition)**.** *Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ be algorithms for a graph problem with average sensitivity $\beta_1, \beta_2, \ldots, \beta_k$, respectively. Let $\mathcal{A}$ be an algorithm that, given a graph G, runs $\mathcal{A}_i$ with probability $\rho_i(G)$ for $i \in [k]$, where $\sum_{i\in[k]} \rho_i(G) = 1$. Let H denote the maximum Hamming weight among the solutions obtained by running $\mathcal{A}$ on G and $\{G^{-e}\}_{e\in E}$. Then the average sensitivity of $\mathcal{A}$ is at most $\sum_{i\in[k]} \rho_i(G) \cdot \beta_i(G) + \mathsf{H} \cdot \mathbb{E}_{e\sim E}\left[\sum_{i\in[k]} |\rho_i(G) - \rho_i(G^{-e})|\right]$.*

Proof. Consider a graph $G = (V, E)$. For a solution S, let $p^G(S)$ denote the probability that S is output on input G by $\mathcal{A}$. Let $p_i^G(S)$ denote the probability that S is output on input G by $\mathcal{A}_i$. For every solution S, we know that $p^G(S) = \sum_{i\in[k]} \rho_i(G) \cdot p_i^G(S)$.

Let $\mathcal{A}(G)$ denote the output distribution of $\mathcal{A}$ on G. Fix $e \in E$. We first bound the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$. In order to transform $\mathcal{A}(G)$ into $\mathcal{A}(G^{-e})$, we first transform $p^G(S)$, for each solution S, into $\sum_{i\in[k]} \rho_i(G) \cdot p_i^{G^{-e}}(S)$. This can be done at a cost of at most $\sum_{i\in[k]} \rho_i(G) \cdot d_{\mathrm{EM}}(\mathcal{A}_i(G), \mathcal{A}_i(G^{-e}))$.

We now convert $\sum_{i\in[k]} \rho_i(G) \cdot p_i^{G^{-e}}(S)$, for each solution S, into $\sum_{i\in[k]} \rho_i(G^{-e}) \cdot p_i^{G^{-e}}(S)$ at a cost of at most $2\mathsf{H} \cdot \frac{1}{2}\sum_{i\in[k]} |\rho_i(G) - \rho_i(G^{-e})|$, where $\frac{1}{2}\sum_{i\in[k]} |\rho_i(G) - \rho_i(G^{-e})|$ is the total variation distance between the probability distributions with which $\mathcal{A}$ selects the algorithms on inputs G and G^{-e}. Hence, the average sensitivity of $\mathcal{A}$ is at most

$$\sum_{i\in[k]} \rho_i(G) \cdot \beta_i(G) + \mathsf{H} \cdot \mathbb{E}_{e\sim E}\left[\sum_{i\in[k]} |\rho_i(G) - \rho_i(G^{-e})|\right]. \qquad \square$$

We separately state the special case of Theorem 7.2.4 for $k = 2$.

Theorem 7.11.2. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two algorithms for a graph problem with average sensitivities $\beta_1(G)$ and $\beta_2(G)$, respectively. Let $\mathcal{A}$ be an algorithm that, given a graph G, runs $\mathcal{A}_1$ with probability $\rho(G)$ and runs $\mathcal{A}_2$ with the remaining probability. Let H denote the maximum Hamming weight among those of solutions obtained by running*

$\mathcal{A}$ on G and $\{G^{-e}\}_{e\in E}$. Then the average sensitivity of $\mathcal{A}$ is at most $\rho(G)\cdot\beta_1(G)+(1-\rho(G))\cdot\beta_2(G)+2\mathsf{H}\cdot\mathbb{E}_{e\sim E}\left[|\rho(G)-\rho(G^{-e})|\right]$.

7.11.4 Locality Implies Low Average Sensitivity

In this section, we prove Theorem 7.2.5, which shows that the existence of an oracle that can simulate access to the solution of a global algorithm $\mathcal{A}$ implies that the average sensitivity of $\mathcal{A}$ is bounded by the query complexity of that oracle.

Theorem 7.2.5 (Locality implies low average sensitivity)**.** *Consider a randomized algorithm $\mathcal{A}:\mathcal{G}\to\mathcal{S}$ for a graph problem, where each solution output by $\mathcal{A}$ is a subset of the set of edges of the input graph. Assume that there exists an oracle $\mathcal{O}$ satisfying the following:*

- *when given access to a graph $G=(V,E)$ and query $e\in E$, the oracle generates a random string $\pi\in\{0,1\}^{r(|V|)}$ and outputs whether e is contained in the solution obtained by running $\mathcal{A}$ on G with π as its random string,*
- *the oracle $\mathcal{O}$ makes at most $q(G)$ queries to G in expectation, where this expectation is taken over the random coins of $\mathcal{A}$ and a uniformly random query $e\in E$.*

Then, $\mathcal{A}$ has average sensitivity at most $q(G)$.

Moreover, given the promise that the input graphs satisfy $|E|\geq|V|$, the statement applies also to algorithms for which each solution is a subset of the vertex set of the input graph.

Proof. We prove the theorem for the case that solutions output by $\mathcal{A}$ are subsets of edges of the input graph. It can be easily modified to work for the case that the solutions output by $\mathcal{A}$ are subsets of vertices of the input graph in which case, we will use the technical condition that $n\leq m$.

Without loss of generality, assume that $\mathcal{A}$ uses $r(n)$ random bits when run on graphs of n vertices[1]. Consider a graph $G = (V, E)$ that $\mathcal{O}$ gets access to. For $e \in E$ and a string $\pi \in \{0,1\}^{r(n)}$, let $Q_{e,\pi}$ denote the set of edges in E queried by $\mathcal{O}$ on input e, while simulating the run of $\mathcal{A}$ with π as the random string. The set $Q_{e,\pi}$ denotes the set of edges e' such that the status of e in the solutions output by $\mathcal{A}$ with randomness π on inputs G and $G^{-e'}$ could be different. For each edge $e' \in E$ and string $\pi \in \{0,1\}^{r(n)}$, define $R_{e',\pi}$ as the set of edges $e \in E$ such that $e' \in Q_{e,\pi}$.

By definition, for each $\pi \in \{0,1\}^{r(n)}$, we have $\sum_{e\in E} |R_{e,\pi}| = \sum_{e\in E} |Q_{e,\pi}|$. Hence we have:

$$\sum_{\pi\in\{0,1\}^{r(n)}} \sum_{e\in E} |R_{e,\pi}| = \sum_{\pi\in\{0,1\}^{r(n)}} \sum_{e\in E} |Q_{e,\pi}|,$$

and

$$\mathbb{E}_{\pi\in\{0,1\}^{r(n)}} \mathbb{E}_{e\sim E} |R_{e,\pi}| \leq \mathbb{E}_{\pi\in\{0,1\}^{r(n)}} \mathbb{E}_{e\sim E} |Q_{e,\pi}| \leq q(G),$$

where the last inequality follows from our assumption on $\mathcal{O}$.

For $\pi \in \{0,1\}^{r(n)}$ and $e \in E$, the set $R_{e,\pi}$ contains the set of edges whose presence in the solution could be affected by the removal of e from G. Therefore, it is a superset of the set of edges contained in the symmetric difference between the outputs of $\mathcal{A}$ on inputs G and G^{-e} when run with π as the random string.

Let $\mathcal{H}_{\mathcal{A},\pi}(G, G')$ denote the Hamming distance between the outputs of the algorithm $\mathcal{A}$ on inputs G and G' when run with π as the random string. As per this notation, for each $e \in E$,

$$\mathbb{E}_{\pi\in\{0,1\}^{r(n)}} \mathcal{H}_{\mathcal{A},\pi}(G, G^{-e}) \leq \mathbb{E}_{\pi\in\{0,1\}^{r(n)}} |R_{e,\pi}|.$$

[1]If $r(G)$ is the length of the random string used for G, we can simply set $r(n) = \max\{r(G) : G = (V, E), |V| = n\}$. If we do not need $r(n)$ bits for some particular graph G on n vertices, we can just throw away the unused bits.

The following claim relates the quantity on the left hand side of the above inequality with the average sensitivity of $\mathcal{A}$.

Claim 7.11.3. *The average sensitivity of $\mathcal{A}$ is bounded as*

$$\beta(G) \leq \mathbb{E}_{e \in E(G)} \mathbb{E}_{\pi \in \{0,1\}^{r(n)}} \mathcal{H}_{\mathcal{A},\pi}(G, G^{-e}).$$

Proof. Fix $G \in \mathcal{G}$ and $e \in E(G)$. We first bound the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$, where $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$ are the output distributions of $\mathcal{A}$ on inputs G and G^{-e}, respectively. For $S \in \mathcal{S}$, let $p_G(S)$ and $p_{G^{-e}}(S)$ denote the probabilities that $\mathcal{A}$ outputs S on G and G^{-e}, respectively. We start with $\mathcal{A}(G)$. Consider a string $\pi \in \{0,1\}^{r(n)}$. Let $S \in \mathcal{S}$ denote the output of $\mathcal{A}$ on input G when using the string π as its random string. Let S' denote the output that is generated when running $\mathcal{A}$ on input G^{-e} with π as the random string. We move a mass of $\frac{1}{2^{r(n)}}$ (corresponding to the string π) from $p_G(S)$ to $p_G(S')$ at a cost of $\frac{d_{\mathrm{Ham}}(S,S')}{2^{r(n)}}$. Moving masses corresponding to every string $\pi \in \{0,1\}^{r(n)}$ this way, we can transform $\mathcal{A}(G)$ to $\mathcal{A}(G^{-e})$. The total cost incurred during this transformation is $\mathbb{E}_{\pi \in \{0,1\}^{r(n)}} \mathcal{H}_{\mathcal{A},\pi}(G, G^{-e})$. Therefore the earth mover's distance between $\mathcal{A}(G)$ and $\mathcal{A}(G^{-e})$ is at most $\mathbb{E}_{\pi \in \{0,1\}^{r(n)}} \mathcal{H}_{\mathcal{A},\pi}(G, G^{-e})$. Therefore the average sensitivity of $\mathcal{A}$ is $\beta(G) \leq \mathbb{E}_{e \in E(G)} \mathbb{E}_{\pi \in \{0,1\}^{r(n)}} \mathcal{H}_{\mathcal{A},\pi}(G, G^{-e})$. $\square$

Therefore, the average sensitivity of $\mathcal{A}$ is:

$$\beta(G) \leq \mathbb{E}_{e \sim E} \mathbb{E}_{\pi \in \{0,1\}^{r(n)}} \mathcal{H}_{\mathcal{A},\pi}(G, G^{-e}) \leq \mathbb{E}_{e \sim E} \mathbb{E}_{\pi \in \{0,1\}^{r(n)}} |R_{e,\pi}| \leq q(G). \qquad \square$$

CHAPTER 8
Erasure-Resilient Graph Property Testing

In this chapter, we define our model of erasure-resilient graph property testing, and present an algorithm and a lower bound for the problem of testing connectedness.

8.1 MODEL DEFINITION AND OUR RESULTS

The complexity of sublinear-time graph algorithms depends on how their inputs are represented. There are two standard ways to represent graphs: adjacency matrices and adjacency lists.

Testing properties of graphs represented by their adjacency matrices was first studied by Goldreich et al. (1998). Adjacency matrices are functional representations of graphs, and hence, erasure-resilient property testing of graphs represented by their adjacency matrices is already covered by our definition of erasure-resilient property testing from Chapter 3. Moreover, for properties that are closed under edge deletions or edge additions, erasure-resilient property testing reduces to standard property testing. Specifically, if a property is closed under edge additions (edge deletions), each erasure in the input adjacency matrix can be replaced with a 1 (0, respectively). Many interesting properties such as connectedness, bipartiteness, colorability, and cycle-freeness are closed under either edge addition or edge deletion and erasure-resiliently testing these properties, therefore, is already resolved.

Property testing of graphs represented by their adjacency lists was defined and studied by Parnas & Ron (2002) (by generalizing the bounded degree graph property testing model, defined and studied by Goldreich & Ron (2002)). Our model is a direct generalization of this well-studied model of graph property testing.

In this model, the study of erasure-resilient property testing raises new conceptual challenges, since adjacency lists are non-functional representations of graphs. Additionally, there is no obvious reduction to standard property testing as in the case of adjacency matrices.

Input representation. We consider simple undirected graphs $G = (V, E)$ represented by their adjacency lists. Some entries in the adjacency lists of a graph G could be erased. Erasures are denoted by $\perp$.

Definition 8.1.1 (α-erased graph)**.** *Let $\alpha \in [0, 1)$ be a parameter. An α-erased graph over a vertex set V is a union of adjacency lists of vertices in V, where at most α fraction of the entries are erased.*

G is a *partially erased* graph over a vertex set V if it is an α-erased graph for some $\alpha \in [0, 1)$. The degree of a vertex $u \in V$, denoted $\deg(u)$, is the length of the adjacency list of u.

Definition 8.1.2 (Completion)**.** *Let $\alpha \in [0, 1)$ be a parameter. A completion of an α-erased graph G is an assignment of vertex labels to erased entries in the adjacency lists of G such that the adjacency lists obtained after the assignment correspond to a graph G'.*

By definition, every partially erased graph has a completion.

Definition 8.1.3 (Nonerased and Half-erased edges)**.** *Let G be a partially erased graph over a vertex set V. For vertices $u, v \in V$, the set $\{u, v\}$ is a nonerased edge in G if u is present in the adjacency list of v and vice versa. The set $\{u, v\}$ is a half-erased edge if u is present in the adjacency list of v but v is not present in the adjacency list of u, or vice versa.*

Given a partially erased graph G, we use m to denote the number of edges in any completion of G, that is, half the sum of lengths of the adjacency lists of all the vertices in G. We call this quantity the number of edges.

Input access. Our algorithms make queries of two types: *degree queries* and *neighbor queries*. A degree query is of the form $v \in V$ and the answer is $\deg(v)$. A neighbor query is of the form (v, i), and the answer is the i^{th} entry in the adjacency list of the vertex v.

Erasure-resilient graph property testing. We investigate property testing on large graphs with missing edges.

Definition 8.1.4. *Let $\alpha \in [0, 1)$, $\varepsilon \in (0, 1)$ be parameters. An α-erased graph G satisfies a property $\mathcal{P}$, if there exists a completion of G that satisfies $\mathcal{P}$. An α-erased graph G is ε-far from a property $\mathcal{P}$, if every completion G' of G is different in at least εm edges from every graph that satisfies $\mathcal{P}$.*

Definition 8.1.5 (Erasure-resilient property tester)**.** *An α-erasure-resilient ε-tester for a property $\mathcal{P}$ gets parameters $\alpha \in [0, 1), \varepsilon \in (0, 1)$ and query access to an α-erased graph G. The tester outputs, with probability at least $2/3$,*

- **accept** *if G satisfies $\mathcal{P}$, and*
- **reject** *if G is ε-far from $\mathcal{P}$.*

The tester has 1-sided error if it always accepts all input graphs G that satisfy $\mathcal{P}$.

Our results. We show that erasure-resilient connectedness testing exhibits a threshold phenomenon. Connectedness can be α-erasure-resiliently ε-tested efficiently if

$\alpha < \varepsilon$. The time complexity of our erasure-resilient testers is identical to their query complexity.

Theorem 8.1.6. *There exists an α-erasure-resilient ε-tester for connectedness of graphs on n vertices and m edges with query and time complexity $O\left(\left(\frac{n}{(\varepsilon-\alpha)m}\right)^3\right)$ that works for every $\varepsilon \in (0, \frac{n}{m})$ and $\alpha \in [0, \varepsilon)$.*

However, the query complexity of erasure-resiliently testing connectedness becomes $\Omega(m)$ as soon as $\alpha \geq \varepsilon$.

Theorem 8.1.7. *For $\varepsilon \in (0, \frac{n}{m})$ and $\alpha \in [\varepsilon, 1)$, at least $\Omega(m)$ queries are necessary to α-erasure-resiliently ε-test connectedness of graphs on m edges.*

8.2 ERASURE-RESILIENTLY TESTING CONNECTEDNESS

In this section, we show upper and lower bounds on the query complexity of erasure-resiliently testing connectedness of graphs.

8.2.1 Characterizing Graphs That Are Far from Connected

In this section, we prove some observations on α-erased graphs that are ε-far from connected. Note that every completion of a patially erased graph on n vertices can be made connected by adding at most $n - 1$ edges. Hence, every partially erased graph on n vertices and m edges is ε-close to connectedness for all $\varepsilon \geq \frac{n}{m}$.

Claim 8.2.1. *Let $\alpha \in [0, 1)$ and $\varepsilon \in (0, \frac{n}{m})$. Let G be an α-erased graph that is ε-far from connected. Then every completion of G has at least $\varepsilon m + 1$ connected components.*

Proof. Assume for the sake of contradiction that there exists a completion G' of G with c connected components such that $c \leq \varepsilon m$. This implies that one can add $c - 1$ edges in order to make G' connected, which is a contradiction. $\square$

Definition 8.2.2 (Small and big subsets)**.** *Given a partially erased graph G and parameters $\varepsilon \in (0, \frac{n}{m})$, $\alpha \in [0, \varepsilon)$, a subset of vertices in G is (ε, α)-small if it has at most $\frac{2n}{(\varepsilon-\alpha)m}$ vertices. It is (ε, α)-big otherwise.*

The *size* of a set is the number of entries in the union of adjacency lists of vertices in that set.

Claim 8.2.3. *Let $\varepsilon \in (0, \frac{n}{m}), \alpha \in [0, \varepsilon)$. Let G be an α-erased graph that is ε-far from connected. Let G' be an arbitrary completion of G. The number of (ε, α)-small connected components in G' is at least $\frac{(\varepsilon+\alpha)m}{2}$.*

Proof. Consider an arbitrary completion G' of G. The number of (ε, α)-big connected components in G' can be at most $\frac{n}{2n/((\varepsilon-\alpha)m)} = \frac{(\varepsilon-\alpha)m}{2}$. By Claim 8.2.1, the total number of connected components in G' is at least $\varepsilon m + 1$. Hence, the number of (ε, α)-small connected components in G' is at least $\frac{(\varepsilon+\alpha)m}{2}$. □

Definition 8.2.4 (Witness to disconnectedness)**.** *Given a partially erased graph G over vertex set V, a set $C \subset V$ is a witness to disconnectedness of G if*

1. *there is at most one erased entry in the union of adjacency lists of vertices in C,*
2. *every nonerased entry in the union of adjacency lists of vertices in C is a vertex from C,*
3. *if there is an erased entry in the adjacency list of $u \in C$, then the degree of u is equal to the number of times u appears as an entry in the union of adjacency lists of vertices in C, and*
4. *if u is present in the adjacency list of v and v is absent from the adjacency list of u, every vertex in C is reachable from v via a BFS.*

Conditions 2, 3 and 4 in Definition 8.2.4 can be restated as:

3. The only erased entry, if any, in the union of adjacency lists of vertices in C is part of a half-erased edge within C.

2. & 4. C forms a single connected component in every completion of G.

Claim 8.2.5. *Let $\varepsilon \in (0, \frac{n}{m}), \alpha \in [0, \varepsilon)$. Let G be an α-erased graph that is ε-far from connected. There are at least $\frac{(\varepsilon-\alpha)m}{2}$ witnesses to the disconnectedness of G. Moreover, the size of each such witness is at most $\left(\frac{2n}{(\varepsilon-\alpha)m}\right)^2$.*

Proof. Consider an arbitrary completion G' of G. By Claim 8.2.3, the number of (ε, α)-small connected components in G' is at least $\frac{(\varepsilon+\alpha)m}{2}$.

Let $C_1, C_2, \dots C_k \subset V$ denote the sets of vertices corresponding to the (ε, α)-small connected components in G'. Since each C_i is an (ε, α)-small connected component in G', the number of edges in the subgraph of G' induced on C_i is at most $\binom{2n/((\varepsilon-\alpha)m)}{2}$. Hence, the size of C_i is at most $\left(\frac{2n}{(\varepsilon-\alpha)m}\right)^2$.

Consider a set C_i. If the union of adjacency lists of vertices in C_i has no erased entries (with respect to G), then C_i is a witness to disconnectedness of G.

Next, assume that the union of adjacency lists of vertices in C_i has exactly one erased entry, where the adjacency lists are with respect to G. Let $u \in C_i$ be the vertex whose adjacency list contains the erased entry. Since C_i is a connected component in G', this erased entry was completed with the label of another vertex $v \in C_i$. This implies that u is present in the adjacency list of v. Moreover, for each nonerased entry v' in the adjacency list of u, the entry u is present in the adjacency list of v'. Hence, the degree of u is equal to the number of times u appears as an entry in the union of adjacency lists of vertices in C_i. Additionally, every vertex in C_i is reachable from v via a BFS, since C_i forms a single connected component in the completion G' and the only erased entry in C_i is in the adjacency list of u (which gets completed by v).

It is also clear that every nonerased entry in the union of adjacency lists of vertices in C_i belong to C_i. Therefore, C_i is a witness to disconnectedness of G.

From the preceding argument, we can see that a set C_i is a *witness to disconnectedness* of G if the union of adjacency lists of vertices in C_i has at most one erased entry, where the adjacency lists are with respect to G.

The total number of erased entries in the adjacency lists of G is at most $\alpha \cdot 2m$. The number of C_i's such that the union of the adjacency lists of vertices in C_i has at least two erased entries is at most $\alpha \cdot m$. Hence, there are at least $\frac{(\varepsilon+\alpha)m}{2} - \alpha m = \frac{(\varepsilon-\alpha)m}{2}$ witnesses to disconnectedness of G with size at most $\left(\frac{2n}{(\varepsilon-\alpha)m}\right)^2$. □

8.2.2 Erasure-Resilient Connectedness Tester

In this section, we describe and analyze a 1-sided error erasure-resilient connectedness tester and prove Theorem 8.1.6. Our tester is specified in Algorithm 8.1.

Algorithm 8.1 Erasure-Resilient Connectedness Tester

```
Input: $\varepsilon \in (0, \frac{n}{m}), \alpha \in [0, \varepsilon)$; oracle access to $\alpha$-erased adjacency lists $G$
1: repeat $\left\lceil \ln 3 \cdot \frac{2n}{(\varepsilon-\alpha)m} \right\rceil$ times:
2:     Sample a vertex $s$ uniformly and independently at random.
3:     Run a BFS starting from $s$ for at most $\left(\frac{2n}{(\varepsilon-\alpha)m}\right)^2$ queries.
4:     if the BFS from Step 3 stops before making $\left(\frac{2n}{(\varepsilon-\alpha)m}\right)^2$ queries then
5:         Reject if the set of vertices reached by the BFS is a witness to disconnectedness of $G$.
6: Accept.
```

Lemma 8.2.6. *Algorithm 8.1 is an α-erasure-resilient ε-tester for connectedness with query complexity $O\left(\left(\frac{n}{(\varepsilon-\alpha)m}\right)^3\right)$ that works for every $\varepsilon \in (0, \frac{n}{m})$ and $\alpha \in [0, \varepsilon)$. Moreover, the time complexity of Algorithm 8.1 is identical to its query complexity.*

Proof. Consider an α-erased graph G over vertex set V. Assume that G is connected. Consider an arbitrary $C \subset V$. Since G is connected, there exists a completion G' of

G such that G' is connected. Therefore, there exists vertices $u \in C$ and $v \in V \setminus C$ such that the adjacency list of u in G' contains v. Hence, C is not a *witness to disconnectedness* of G. Therefore, the tester accepts G.

Next, assume that G is ε-far from connected. Let B denote the set of all (ε, α)-small witnesses to disconnectedness of G. Let $C \subseteq V$ be an element of B. There exists at least one vertex $x \in C$ such that a BFS starting from x reaches every vertex in C. If the union of adjacency lists of vertices in C does not contain any erased entries, then every vertex in C is reachable from every other vertex in C. Otherwise, condition 3 of Definition 8.2.4 guarantees the existence of such a vertex. If Algorithm 8.1 samples such a vertex, it will detect a witness to the disconnectedness of G and reject.

Therefore, the probability that the Algorithm 8.1 rejects in a single iteration is at least $\frac{|B|}{n} \geq \frac{(\varepsilon-\alpha)m}{2n}$. Hence, the probability that Algorithm 8.1 accepts is at most

$$\left(1 - \frac{(\varepsilon - \alpha)m}{2n}\right)^{\left\lceil \ln 3 \cdot \frac{2n}{(\varepsilon-\alpha)m} \right\rceil} < \exp(-\ln 3) = \frac{1}{3}.$$

Thus, Algorithm 8.1 rejects with probability at least $\frac{2}{3}$.

The number of queries made by Algorithm 8.1 in Step 3 is $O\left(\left(\frac{n}{(\varepsilon-\alpha)m}\right)^2\right)$. Thus, the query complexity of Algorithm 8.1 is $O\left(\left(\frac{n}{(\varepsilon-\alpha)m}\right)^3\right)$.

Checking (in Step 5) whether a set C is a witness to disconnectedness of G can be done by making a constant number of passes over the union of adjacency lists of vertices in C. Since we perform this check only for sets C of size $O\left(\left(\frac{n}{(\varepsilon-\alpha)m}\right)^2\right)$, the time complexity of Algorithm 8.1 is $O\left(\left(\frac{n}{(\varepsilon-\alpha)m}\right)^3\right)$. □

8.2.3 Erasure-Resilient Connectedness Testing: A Lower Bound

In this section, we prove Theorem 8.1.7.

Proof of Theorem 8.1.7. We apply Yao's minimax principle (as stated by Raskhodnikova & Smith (2006)). Specifically, we construct distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$, the former over connected graphs and the latter over graphs that are ε-far from connected, such that every deterministic α-erasure-resilient ε-tester for connectedness makes $\Omega(m)$ queries to distinguish the two distributions.

Fix $t \in \mathbb{N}$. Let k be an even number and let $n = kt + 1$. We first construct two partially erased graphs G_1 and G_2, containing n vertices each. The vertex sets of both G_1 and G_2 are partitioned into $k + 1$ parts. Each of the first k parts has t vertices which induce a cycle. In each part, there is exactly one vertex with degree 3. Its adjacency list contains its neighbors on the cycle and one $\perp$. The last part contains a single vertex $v^\star$. In G_1, the adjacency list of vertex $v^\star$ has length k and contains the labels of the degree-3 vertices in the cycles. In G_2, the vertex $v^\star$ is isolated; that is, its adjacency list is empty. We mention that $v^\star$ is simply a shorthand to denote the vertex belonging to this single-vertex part and is *not* the label of that vertex. See Figure 8.1 for a representation of G_1 and G_2.

The fraction of erased entries in the adjacency lists of G_1 and G_2 are $\frac{1}{2t+2}$ and $\frac{1}{2t+1}$, respectively. That is, G_1 and G_2 are both α-erased graphs for $\alpha = \frac{1}{2t+1}$.

We can obtain a connected completion of G_1 by connecting the vertex $v^\star$ to the degree-3 vertices in the k connected components. However, at least $\frac{k}{2}$ edges need to be added to every completion of G_2 to make it connected. Hence, G_2 is ε-far from connected for every $\varepsilon \leq \frac{k/2}{kt+k/2} = \frac{1}{2t+1} = \alpha$.

Description of distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$. We define the two distributions $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ as uniform distributions over the set of all partially erased graphs isomorphic to G_1 and G_2, respectively. Each partially erased graph sampled from $\mathcal{D}_{\text{yes}}$ is connected. Each partially erased graph sampled from $\mathcal{D}_{\text{no}}$ is ε-far from connected for all $\varepsilon \leq \alpha$.

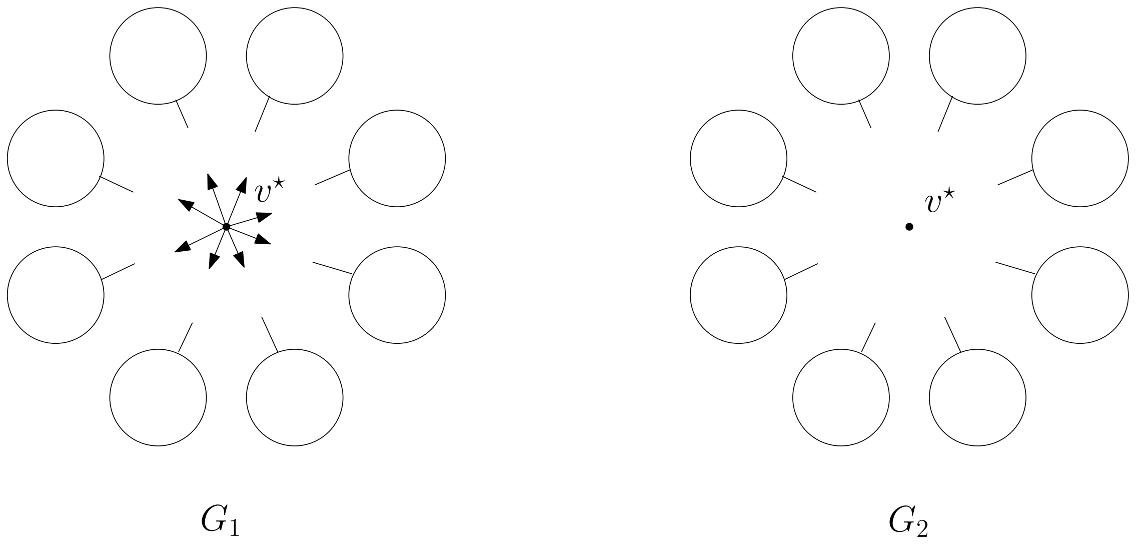

Figure 8.1: The partially erased graphs G_1 and G_2. The lines without arrows represent erased entries in the adjacency lists of corresponding vertices. The lines with arrows indicate that the entry corresponds to the vertex to which the arrow points to.

Claim 8.2.7. *Every deterministic algorithm A has to make $\Omega(kt)$ queries to distinguish $\mathcal{D}_{yes}$ and $\mathcal{D}_{no}$ with probability at least $2/3$.*

Proof. We make Definition 8.2.8 to ease the exposition of the proof.

Definition 8.2.8 (Queried, seen, and unknown vertices)**.** *Given a particular execution of A, we say that a vertex v is (1) queried if A has made some query about the adjacency list of v, and (2) seen if A has received v as an answer to one of its queries but never queried the adjacency list of v. The vertices that are neither queried nor seen are unknown.*

Let q denote the number of queries made by A. The vertex $v^\star$ is *unknown* before A makes its first query. It is impossible for the status of $v^\star$ to become *seen*, since $v^\star$ is not connected to any other vertex in the partially erased graphs sampled from either $\mathcal{D}_{\text{yes}}$ or $\mathcal{D}_{\text{no}}$. Let E_i for $i \in [q]$ denote the event that $v^\star$ is *queried* for the first time in the i-th query. The event E_i occurs if the algorithm queries an *unknown* vertex in its i-th query and that vertex happens to be $v^\star$. Therefore, the probability of E_i is at most $1/(kt+1-i)$.

We now calculate the probability of the event that $v^\star$ is a *queried* vertex by the end of an execution of A, where the probability is taken over the randomness in the distribution from which the partially erased graph is sampled. By the union bound, the probability of this event is at most

$$\sum_{i \in [q]} \Pr\left[E_i\right] \leq \frac{q}{kt+1-q} = \frac{1}{7} < \frac{1}{6},$$

for $q = \frac{kt+1}{8}$.

If $v^\star$ is not a *queried* vertex by the end of a particular execution, then the view of the partially erased graph obtained by A in that execution could have been generated from a partially erased graph sampled according to either $\mathcal{D}_{\text{yes}}$ or $\mathcal{D}_{\text{no}}$ with equal probability. Therefore, an execution of A cannot distinguish $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ with probability more than $1/2$, conditioned on $v^\star$ not being a *queried* vertex in that execution. Therefore, the probability that A distinguishes $\mathcal{D}_{\text{yes}}$ and $\mathcal{D}_{\text{no}}$ is at most $\frac{1}{2} + \frac{1}{7} < \frac{2}{3}$. $\square$

Note that $m \leq kt + k$. Thus, every algorithm that uses only degree and neighbor queries has to make $\Omega(m)$ queries to erasure-resiliently test connectedness. $\square$

CHAPTER 9
Conclusions and Open Problems

The key contributions of this thesis involve developing new models and metrics to reason about algorithms that analyze large datasets containing missing entries. We believe that this is an important step towards enhancing the understanding of computation in the presence of large faulty datasets. Our work leaves open several directions for future research in this field. One of the most important directions is to expand the study of *erasure-resilience* to other models of sublinear algorithms such as streaming and sketching algorithms (Alon et al. (1999)), local algorithms (Rubinfeld et al. (2011)) and massively parallel algorithms (Karloff et al. (2010)). We list more specific open questions below.

Adaptivity in erasure-resilient testing. Most of our efficient (and in some cases, optimal) erasure-resilient testers from Chapter 3 are adaptive. Examples include our testers for monotonicity and convexity over the line, $[n]$. Designing nonadaptive erasure-resilient testers for these and other important properties is an interesting open question.

Better erasure-resilient monotonicity testers for hypergrid domains. The fraction of erasures that our erasure-resilient monotonicity tester (Chapter 3) for hypergrid domains, $[n]^d$, can tolerate decreases inversely with the dimension d. We also show that an inverse dependence on $\sqrt{d}$ is necessary for all testers that work by sampling axis-parallel lines uniformly at random and then testing monotonicity on them. It is an interesting combinatorial question to determine, for a function that is far from monotone, the exact tradeoff between the fraction of erasures and the fraction of axis parallel lines that are far from monotone. Another direction is to design better erasure-resilient monotonicity testers for the case of the hypercube, $[2]^d$; the optimal standard monotonicity testers for this case do not rely on the axis-parallel line strategy (Khot et al. (2018)).

Local list decoding from erasures. Another important open question raised by our work is whether local list decoding is significantly easier in terms of the query complexity, the list size, or the rate of codes when corruptions are in the form of erasures. The same question can be asked about approximate local list decoding. Our local list erasure-decoder for the Hadamard code shows that there is some advantage for having erasures over errors, in terms of the list size and query complexity, for some settings of parameters. A positive or negative answer to this question, combined with our result on the equivalence of errors and erasures in the local decoding regime, will enhance the understanding of whether local list decoding is an inherently more powerful model when compared to local decoding.

Erasure-resilient testing in different erasure models. In this work, we discussed both the oblivious as well as the semi-oblivious adversarial model of erasure generation. A natural question to study is whether our semi-oblivious linearity tester is optimal. Other questions include testing linearity (and other properties) against more general b-semi-oblivious adversaries that can erase at most b points before seeing each query of the tester, where $b \in \mathbb{N}$. Moving away from adversarial erasure oracles, one can also consider oracles that generate erasures according to some distribution and study erasure-resilient property testing in those models.

Average sensitivity bounds for other optimization problems. In Chapter 7, our focus was exclusively on designing stable-on-average algorithms for optimization problems concerning graphs. However, our definition of average sensitivity extends to algorithms for most optimization problems. It is thus natural to ask whether one can design stable-on-average approximation algorithms for other optimization problems.

BIBLIOGRAPHY

Ailon, N., & Chazelle, B. (2006). Information theory in property testing and monotonicity testing in higher dimension. *Information and Computation*, *204*(11), 1704–1717.

Ailon, N., Chazelle, B., Comandur, S., & Liu, D. (2007). Estimating the distance to a monotone function. *Random Structures & Algorithms*, *31*(3), 371–383.

Alon, N., Matias, Y., & Szegedy, M. (1999). The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, *58*(1), 137–147.

Arora, S., Lund, C., Motwani, R., Sudan, M., & Szegedy, M. (1998). Proof verification and intractability of approximation problems. *Journal of the ACM*, *45*(3), 501–555.

Arora, S., & Safra, S. (1998). Probabilistic checkable proofs: A new characterization of NP. *Journal of the ACM*, *45*(1), 70–122.

Awasthi, P., Jha, M., Molinaro, M., & Raskhodnikova, S. (2016). Testing Lipschitz functions on hypergrid domains. *Algorithmica*, *74*(3), 1055–1081.

Babai, L., Fortnow, L., Levin, L. A., & Szegedy, M. (1991). Checking computations in polylogarithmic time. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) 1991*, (pp. 21–31).

Babai, L., Fortnow, L., Nisan, N., & Wigderson, A. (1993). BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, *3*(4), 307–318.

Balcan, M., Blais, E., Blum, A., & Yang, L. (2012). Active property testing. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2012*, (pp. 21–30).

Batu, T., Fortnow, L., Rubinfeld, R., Smith, W. D., & White, P. (2013). Testing closeness of discrete distributions. *Journal of the ACM*, *60*(1), 4:1–4:25.

Batu, T., Rubinfeld, R., & White, P. (2005). Fast approximate PCPs for multidimensional bin-packing problems. *Information and Computation*, *196*(1), 42–56.

Bavelas, A. (1950). Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, *22*(6), 725–730.

Beauchamp, M. A. (1965). An improved index of centrality. *Behavioral Science*, *10*(2), 161–163.

Beimel, A., Ishai, Y., Kushilevitz, E., & Raymond, J.-F. (2002). Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2002*, (pp. 261–270).

Bellare, M., Coppersmith, D., Håstad, J., Kiwi, M. A., & Sudan, M. (1996). Linearity testing in characteristic two. *IEEE Transcations on Information Theory*, *42*(6), 1781–1795.

Belovs, A. (2018). Adaptive lower bound for testing monotonicity on the line. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018*, (pp. 31:1–31:10).

Belovs, A., & Blais, E. (2015). Quantum algorithm for monotonicity testing on the hypercube. *Theory of Computing*, *11*, 403–412.

Belovs, A., & Blais, E. (2016). A polynomial lower bound for testing monotonicity. In *Proceedings of the ACM SIGACT Symposium on Theory of Computing (STOC) 2016*, (pp. 1021–1032).

Ben-Aroya, A., Efremenko, K., & Ta-Shma, A. (2010a). Local list decoding with a constant number of queries. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2010*, (pp. 715–722).

Ben-Aroya, A., Efremenko, K., & Ta-Shma, A. (2010b). A note on amplifying the error-tolerance of locally decodable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, *17*, 134.

Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., & Vadhan, S. P. (2006). Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, *36*(4), 889–974.

Ben-Sasson, E., Harsha, P., & Raskhodnikova, S. (2005). Some 3CNF properties are hard to test. *SIAM Journal on Computing*, *35*(1), 1–21.

Berman, P., Murzabulatov, M., & Raskhodnikova, S. (2016a). The power and limitations of uniform samples in testing properties of figures. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS) 2016*, (pp. 45:1–45:14).

Berman, P., Murzabulatov, M., & Raskhodnikova, S. (2016b). Testing convexity of figures under the uniform distribution. In *Proceedings of the International Symposium on Computational Geometry (SoCG) 2016*, (pp. 17:1–17:15).

Berman, P., Murzabulatov, M., & Raskhodnikova, S. (2016c). Tolerant testers of image properties. In *Proceedings of the International Colloquium on Automata, Languages, and Programming, ICALP 2016*, (pp. 90:1–90:14).

Bhattacharyya, A., Grigorescu, E., Jung, K., Raskhodnikova, S., & Woodruff, D. P. (2012). Transitive-closure spanners. *SIAM Journal on Computing*, *41*(6), 1380–1425.

Black, H., Chakrabarty, D., & Seshadhri, C. (2018). A $o(d) \cdot \text{polylog}\, n$ monotonicity tester for Boolean functions over the hypergrid $[n]^d$. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, (SODA) 2018*, (pp. 2133–2151).

Blais, E., Brody, J., & Matulef, K. (2012). Property testing lower bounds via communication complexity. *Computational Complexity*, *21*(2), 311–358.

Blais, E., Raskhodnikova, S., & Yaroslavtsev, G. (2014). Lower bounds for testing properties of functions over hypergrid domains. In *Proceedings of the IEEE Conference on Computational Complexity (CCC) 2014*, (pp. 309–320).

Blinovsky, V. M. (1986). Bounds for codes in the case of list decoding of finite volume. *Problems of Information Transmission*, *22*(1), 7–19.

Blum, M., Luby, M., & Rubinfeld, R. (1993). Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, *47*(3), 549–595.

Bousquet, O., & Elisseeff, A. (2002). Stability and generalization. *Journal of Machine Learning Research*, *2*, 499–526.

Briët, J., Chakraborty, S., García-Soriano, D., & Matsliah, A. (2012). Monotonicity testing and shortest-path routing on the cube. *Combinatorica*, *32*(1), 35–53.

Cai, J., Pavan, A., & Sivakumar, D. (1999). On the hardness of permanent. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science (STACS) 1999*, vol. 1563, (pp. 90–99).

Canonne, C. L., Grigorescu, E., Guo, S., Kumar, A., & Wimmer, K. (2017). Testing k-monotonicity. In *Proceedings of the Innovations in Theoretical Computer Science Conference, (ITCS) 2017*, (pp. 29:1–29:21).

Chakrabarty, D., Dixit, K., Jha, M., & Seshadhri, C. (2017). Property testing on product distributions: Optimal testers for bounded derivative properties. *ACM Transactions on Algorithms*, *13*(2), 20:1–20:30.

Chakrabarty, D., & Seshadhri, C. (2013). Optimal bounds for monotonicity and Lipschitz testing over hypercubes and hypergrids. In *Proceedings of the ACM Symposium on Theory of Computing (STOC) 2013*, (pp. 419–428).

Chakrabarty, D., & Seshadhri, C. (2014). An optimal lower bound for monotonicity testing over hypergrids. *Theory of Computing*, *10*, 453–464.

Chakrabarty, D., & Seshadhri, C. (2016). An $o(n)$ monotonicity tester for Boolean functions over the hypercube. *SIAM Journal on Computing*, *45*(2), 461–472.

Chakrabarty, D., & Seshadhri, C. (2019). Adaptive Boolean monotonicity testing in total influence time. In *Proceedings of the Innovations in Theoretical Computer Science Conference, (ITCS) 2019*, (pp. 20:1–20:7).

Chen, X., De, A., Servedio, R. A., & Tan, L. (2015). Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) 2015*, (pp. 519–528).

Chen, X., Servedio, R. A., & Tan, L. (2014). New algorithms and lower bounds for monotonicity testing. In *Proceedings of IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2014*, (pp. 286–295).

Chor, B., Kushilevitz, E., Goldreich, O., & Sudan, M. (1998). Private information retrieval. *Journal of the ACM*, *45*(6), 965–981.

Devroye, L. (1986). A note on the height of binary search trees. *Journal of the ACM*, *33*(3), 489–498.

Dinur, I. (2007). The PCP theorem by gap amplification. *Journal of the ACM*, *54*(3), 12.

Dinur, I., & Reingold, O. (2006). Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM Journal on Computing*, *36*(4), 975–1024.

Dixit, K., Jha, M., Raskhodnikova, S., & Thakurta, A. (2013). Testing the Lipschitz property over product distributions with applications to data privacy. In *Proceedings of the Theory of Cryptography Conference (TCC) 2013*, (pp. 418–436).

Dixit, K., Raskhodnikova, S., Thakurta, A., & Varma, N. M. (2018). Erasure-resilient property testing. *SIAM Journal on Computing*, *47*(2), 295–329.

Dodis, Y., Goldreich, O., Lehman, E., Raskhodnikova, S., Ron, D., & Samorodnitsky, A. (1999). Improved testing algorithms for monotonicity. In *Third International Workshop on Randomization and Approximation Techniques in Computer Science, and Second International Workshop on Approximation Algorithms for Combinatorial Optimization Problems RANDOM-APPROX'99, 1999, Proceedings*, (pp. 97–108).

Drmota, M. (2003). An analytic approach to the height of binary search trees II. *Journal of the ACM*, *50*(3), 333–374.

Dvir, Z., Gopalan, P., & Yekhanin, S. (2011). Matching vector codes. *SIAM Journal on Computing*, *40*(4), 1154–1178.

Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Theory of Cryptography Conference (TCC) 2006*, (pp. 265–284).

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, *17*, 449–467.

Efremenko, K. (2012). 3-query locally decodable codes of subexponential length. *SIAM Journal on Computing*, *41*(6), 1694–1703.

Erdős, P., & Rényi, A. (1959). On random graphs. *Publicationes Mathematicae*, *6*, 290–297.

Ergün, F., Kannan, S., Kumar, R., Rubinfeld, R., & Viswanathan, M. (2000). Spot-checkers. *Journal of Computer and System Sciences*, *60*(3), 717–751.

Ergün, F., Kumar, R., & Rubinfeld, R. (2004). Fast approximate probabilistically checkable proofs. *Information and Computation*, *189*(2), 135–159.

Fattal, S., & Ron, D. (2007). Approximating the distance to convexity. Unpublished manuscript. Uploaded at http://www.eng.tau.ac.il/∼danar/Public-pdf/app-conv.pdf.

Fattal, S., & Ron, D. (2010). Approximating the distance to monotonicity in high dimensions. *ACM Transactions on Algorithms*, *6*(3), 52:1–52:37.

Fischer, E. (2004). On the strength of comparisons in property testing. *Information and Computation*, *189*(1), 107–116.

Fischer, E., & Fortnow, L. (2006). Tolerant versus intolerant testing for Boolean properties. *Theory of Computing*, *2*(9), 173–183.

Fischer, E., Lachish, O., & Vasudev, Y. (2015). Trading query complexity for sample-based testing and multi-testing scalability. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2015*, (pp. 1163–1182).

Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., & Samorodnitsky, A. (2002). Monotonicity testing over general poset domains. In *Proceedings of the ACM Symposium on Theory of Computing (STOC) 2002*, (pp. 474–483).

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, *40*(1), 35–41.

Fubini, G. (1907). Sugli integrali multipli. *Roma Accademia dei Lincei Rendiconti (5)*, *16*(1), 608–614.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Gemmell, P., Lipton, R. J., Rubinfeld, R., Sudan, M., & Wigderson, A. (1991). Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) 1991*, (pp. 32–42).

Gemmell, P., & Sudan, M. (1992). Highly resilient correctors for polynomials. *Information Processing Letters*, *43*(4), 169–174.

Goldreich, Rubinfeld, & Sudan (2000a). Learning polynomials with queries: The highly noisy case. *SIAM Journal on Discrete Mathematics*, *13*.

Goldreich, O. (2011a). A brief introduction to property testing. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, (pp. 465–469). Springer.

Goldreich, O. (2011b). Introduction to testing graph properties. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, (pp. 470–506). Springer.

Goldreich, O. (2011c). Short locally testable codes and proofs. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, (pp. 333–372). Springer.

Goldreich, O. (2017). *Introduction to Property Testing*. Cambridge University Press.

Goldreich, O., Goldwasser, S., Lehman, E., Ron, D., & Samorodnitsky, A. (2000b). Testing monotonicity. *Combinatorica*, *20*(3), 301–337.

Goldreich, O., Goldwasser, S., & Ron, D. (1998). Property testing and its connection to learning and approximation. *Journal of the ACM*, *45*(4), 653–750.

Goldreich, O., & Kaufman, T. (2011). Proximity oblivious testing and the role of invariances. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011*, (pp. 579–592).

Goldreich, O., & Levin, L. A. (1989). A hard-core predicate for all one-way functions. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) 1989*, (pp. 25–32).

Goldreich, O., & Ron, D. (2002). Property testing in bounded degree graphs. *Algorithmica*, *32*(2), 302–343.

Goldreich, O., & Ron, D. (2011). On proximity-oblivious testing. *SIAM Journal on Computing*, *40*(2), 534–566.

Goldreich, O., & Ron, D. (2016). On sample-based testers. *ACM Transactions on Computation Theory*, *8*(2), 7:1–7:54.

Goldreich, O., & Shinkar, I. (2016). Two-sided error proximity oblivious testing. *Random Structures & Algorithms*, *48*(2), 341–383.

Gopi, S., Kopparty, S., Oliveira, R., Ron-Zewi, N., & Saraf, S. (2018). Locally testable and locally correctable codes approaching the Gilbert-Varshamov bound. *IEEE Transactions on Information Theory*, *64*(8).

Grinberg, A., Shaltiel, R., & Viola, E. (2018). Indistinguishability by adaptive procedures with advice, and lower bounds on hardness amplification proofs. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2018*, (pp. 956–966).

Guo, A., & Kopparty, S. (2016). List-decoding algorithms for lifted codes. *IEEE Transcations on Information Theory*, *62*(5), 2719–2725.

Gupta, A., Ligett, K., McSherry, F., Roth, A., & Talwar, K. (2010). Differentially private combinatorial optimization. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, (SODA) 2010*, (pp. 1106–1125).

Guruswami, V. (2003). List decoding from erasures: bounds and code constructions. *IEEE Transactions on Information Theory*, *49*(11), 2826–2833.

Guruswami, V., & Indyk, P. (2005). Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transcations on Information Theory*, *51*(10), 3393–3400.

Guruswami, V., & Rudra, A. (2005). Tolerant locally testable codes. In *8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th International Workshop on Randomization and Computation, RANDOM 2005, Proceedings*, (pp. 306–317).

Guruswami, V., & Vadhan, S. P. (2010). A lower bound on list size for list decoding. *IEEE Transcations on Information Theory*, *56*(11), 5681–5688.

Gutfreund, D., & Rothblum, G. N. (2008). The complexity of local list decoding. In *11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Proceedings*, (pp. 455–468).

Halevy, S., & Kushilevitz, E. (2008). Testing monotonicity over graph products. *Random Structures & Algorithms*, *33*(1), 44–67.

Hay, M., Li, C., Miklau, G., & Jensen, D. D. (2009). Accurate estimation of the degree distribution of private networks. In *Proceedings of the IEEE International Conference on Data Mining (ICDM) 2009*, (pp. 169–178).

Hemenway, B., Ron-Zewi, N., & Wootters, M. (2017). Local list recovery of high-rate tensor codes & applications. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2017*. IEEE Computer Society.

Huber, P. J. (2011). *Robust statistics*. Springer.

Impagliazzo, R., Jaiswal, R., Kabanets, V., & Wigderson, A. (2010). Uniform direct product theorems: Simplified, optimized, and derandomized. *SIAM Journal on Computing*, *39*(4), 1637–1665.

Jha, M., & Raskhodnikova, S. (2013). Testing and reconstruction of Lipschitz functions with applications to data privacy. *SIAM Journal on Computing*, *42*(2), 700–731.

Karger, D. R. (1993). Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA) 1993*, (pp. 21–30).

Karloff, H. J., Suri, S., & Vassilvitskii, S. (2010). A model of computation for MapReduce. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) 2010*, (pp. 938–948).

Karwa, V., Raskhodnikova, S., Smith, A. D., & Yaroslavtsev, G. (2014). Private analysis of graph structure. *ACM Transactions on Database Systems*, *39*(3), 22:1–22:33.

Karwa, V., & Slavkovic, A. B. (2012). Differentially private graphical degree sequences and synthetic graphs. In *Privacy in Statistical Databases - UNESCO Chair in Data Privacy, International Conference (PSD) 2012*, (pp. 273–285).

Kasiviswanathan, S. P., Nissim, K., Raskhodnikova, S., & Smith, A. D. (2013). Analyzing graphs with node differential privacy. In *Proceedings of the Theory of Cryptography Conference (TCC) 2013*, (pp. 457–476).

Katz, J., & Trevisan, L. (2000). On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the ACM Symposium on Theory of Computing (STOC) 2000*, (pp. 80–86).

Kearns, M. J., & Li, M. (1993). Learning in the presence of malicious errors. *SIAM Journal on Computing*, *22*(4), 807–837.

Kearns, M. J., & Ron, D. (2000). Testing problems with sublearning sample complexity. *Journal of Computer and System Sciences*, *61*(3), 428–456.

Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) 2003*, (pp. 137–146).

Khot, S., Minzer, D., & Safra, M. (2018). On monotonicity testing and boolean isoperimetric-type theorems. *SIAM Journal on Computing*, *47*(6), 2238–2276.

Khot, S. A., & Regev, O. (2003). Vertex cover might be hard to approximate to within $2 - \epsilon$. In *Proceedings of the Annual IEEE Conference on Computational Complexity (CCC) 2003*, (pp. 379–386).

Kopparty, S. (2015). List-decoding multiplicity codes. *Theory of Computing*, *11*, 149–182.

Kopparty, S., & Saraf, S. (2013). Local list-decoding and testing of random linear codes from high error. *SIAM Journal on Computing*, *42*(3), 1302–1326.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, *7*(1), 48–50.

Kushilevitz, E., & Mansour, Y. (1993). Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, *22*(6), 1331–1348.

Lehman, E., & Ron, D. (2001). On disjoint chains of subsets. *Journal of Combinatorial Theory, Series A*, *94*(2), 399–404.

Levi, A., Pallavoor, R. K. S., Raskhodnikova, S., & Varma, N. (2019). Erasure-resilient sublinear-time algorithms for graphs. *Unpublished manuscript.*

Lipton, R. J. (1990). Efficient checking of computations. In *Proceedings of the Annual ACM Symposium on Theoretical Aspects of Computer Science (STACS) 1990*, (pp. 207–215).

Lipton, R. J. (1991). New directions in testing. In *Distributed Computing and Cryptography*, vol. 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, (pp. 191–202). DIMACS/AMS.

Marchiori, M., & Latora, V. (2000). Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, *285*(3-4), 539–546.

Meir, O. (2014). Combinatorial PCPs with efficient verifiers. *Computational Complexity*, *23*(3), 355–478.

Meir, O. (2016). Combinatorial PCPs with short proofs. *Computational Complexity*, *25*(1), 1–102.

Meulemans, W., Speckmann, B., Verbeek, K., & Wulms, J. (2018). A framework for algorithm stability and its application to kinetic euclidean MSTs. In *Proceedings of the Latin American Symposium on Theoretical Informatics (LATIN) 2018*, (pp. 805–819).

Murai, S., & Yoshida, Y. (2019). Sensitivity analysis of centralities on unweighted networks. In *Proceedings of The World Wide Web Conference, WWW 2019*, (pp. 1332–1342).

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E*, *69*(6), 066133.

Newman, M. E. J. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, *103*(23), 8577–8582.

Nguyen, H. N., & Onak, K. (2008). Constant-time approximation algorithms via local improvements. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS) 2008*, (pp. 327–336).

Nissim, K., Raskhodnikova, S., & Smith, A. D. (2007). Smooth sensitivity and sampling in private data analysis. In *Proceedings of the ACM Symposium on Theory of Computing (STOC) 2007*, (pp. 75–84).

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab.

Pallavoor, R. K. S., Raskhodnikova, S., & Varma, N. M. (2018). Parameterized property testing of functions. *ACM Transactions on Computation Theory*, *9*(4), 17:1–17:19.

Parnas, M., & Ron, D. (2002). Testing the diameter of graphs. *Random Structures & Algorithms*, *20*(2), 165–183.

Parnas, M., Ron, D., & Rubinfeld, R. (2003). On testing convexity and submodularity. *SIAM Journal on Computing*, *32*(5), 1158–1184.

Parnas, M., Ron, D., & Rubinfeld, R. (2006). Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, *6*(72), 1012–1042.

Pitt, L. (1985). A simple probabilistic approximation algorithm for vertex cover. Tech. rep., Yale University.

Pittel, B. (1984). On growing random binary trees. *Journal of Mathematical Analysis and Applications*, *103*(2), 461 – 480.

Polishchuk, A., & Spielman, D. A. (1994). Nearly-linear size holographic proofs. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC) 1994*, (pp. 194–203).

Raskhodnikova, S., Ron-Zewi, N., & Varma, N. M. (2019). Erasures vs. errors in local decoding and property testing. In *Proceedings of the Innovations in Theoretical Computer Science Conference, (ITCS) 2019*, (pp. 63:1–63:21).

Raskhodnikova, S., & Smith, A. D. (2006). A note on adaptivity in testing properties of bounded degree graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, *13*(089).

Raskhodnikova, S., & Smith, A. D. (2016). Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS) 2016*, (pp. 495–504).

Reed, B. A. (2003). The height of a random binary search tree. *Journal of the ACM*, *50*(3), 306–332.

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

Rubinfeld, R., & Blais, E. (2016). Something for (almost) nothing: New advances in sublinear-time algorithms. In *Handbook of Big Data.*, (pp. 155–167). Chapman and Hall/CRC.

Rubinfeld, R., & Sudan, M. (1996). Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, *25*(2), 252–271.

Rubinfeld, R., Tamir, G., Vardi, S., & Xie, N. (2011). Fast local computation algorithms. In *Innovations in Computer Science - ICS 2010. Proceedings*, (pp. 223–238).

Sabidussi, G. (1966). The centrality index of a graph. *Psychometrika*, *31*(4), 581–603.

Saks, M. E., & Seshadhri, C. (2017). Estimating the longest increasing sequence in polylogarithmic time. *SIAM Journal on Computing*, *46*(2), 774–823.

Shalev-Shwartz, S., & Ben-David, S. (2009). *Understanding Machine Learning*. From Theory to Algorithms. Cambridge: Cambridge University Press.

Spencer, J., & Florescu, L. (2014). *Asymptopia*. American Mathematical Society.

Sudan, M., Trevisan, L., & Vadhan, S. P. (2001). Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, *62*(2), 236–266.

Trevisan, L. (2003). List-decoding using the XOR lemma. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS) 2003*, (pp. 126–135).

Trevisan, L. (2004). Some applications of coding theory in computational complexity. *Computing Research Repository (CoRR)*, *cs.CC/0409044*.

Varma, N., & Yoshida, Y. (2019). Average sensitivity of graph algorithms. *Computing Research Repository (CoRR)*, *abs/1904.03248*.

Yekhanin, S. (2008). Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM*, *55*(1), 1:1–1:16.

Yekhanin, S. (2012). Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, *6*(3), 139–255.

Yoshida, Y., Yamamoto, M., & Ito, H. (2012). Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM Journal on Computing*, *41*(4), 1074–1093.

CURRICULUM VITAE

Nithin Varma, M.Sc.

2019-09-25

109 Colborne Road	MCS 217
Brighton, MA 02135	Boston University
(814) 574 7030	Boston, MA 02215
nithvarma@gmail.com	nvarma@bu.edu

Academic Training:

09/2019(expected)	PhD Boston University, Boston, MA; Computer Science
07/2014	MSc TIFR Mumbai, India; Computer Science
07/2011	BTech NIT Calicut, Kerala, India; Computer Science and Engineering

Doctoral Research:

Title:	Analyzing Massive Datasets with Missing Entries: Models and Algorithms
Thesis advisor:	Dr. Sofya Raskhodnikova, PhD
Defense date:	July 10, 2019
Summary:	In this work, we propose novel computational models for algorithms to analyze massive datasets containing missing (or *erased*) entries. Further, we design efficient algorithms in our models for various well-studied problems, and investigate the relationship of our models to the existing models of algorithms for big data.

Original, Peer Reviewed Publications (newest first):

1. Bipartite Graphs of Small Readability.
   Rayan Chikhi, Vladan Jovičič, Stefan Kratsch, Paul Medvedev, Martin Milanič, Sofya Raskhodnikova, Nithin Varma.
   Elsevier *Theoretical Computer Science*
   In press, journal pre-proof, Available online July 2019.

2. Erasures vs. Errors in Local Decoding and Property Testing.
   Sofya Raskhodnikova, Noga Ron-Zewi, Nithin Varma.
   Proceedings of ITCS 2019: 63:1-63:21.

3. Brief Announcement: Erasure-Resilience versus Tolerance to Errors.
   Sofya Raskhodnikova, Nithin Varma.
   Proceedings of ICALP 2018: 111:1-111:3.

4. Bipartite Graphs of Small Readability.
   Rayan Chikhi, Vladan Jovičić, Stefan Kratsch, Paul Medvedev, Martin Milanič, Sofya Raskhodnikova, Nithin Varma.
   Proceedings of COCOON 2018: 467-479.

5. Erasure-Resilient Property Testing.
   Kashyap Dixit, Sofya Raskhodnikova, Abhradeep Thakurta, Nithin Varma.
   SIAM *Journal on Computing, 47(2):295–329, 2018.*

6. Parameterized Function Property Testing.
   Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, Nithin Varma.
   ACM *Transactions of Computation Theory*, 9(4): 17:1-17:19, 2018.

7. Parameterized Function Property Testing.
   Ramesh Krishnan S. Pallavoor, Sofya Raskhodnikova, Nithin Varma.
   Proceedings of ITCS 2017: 12:1-12:17.

8. Erasure-Resilient Property Testing.
   Kashyap Dixit, Sofya Raskhodnikova, Abhradeep Thakurta, Nithin Varma.
   ICALP 2016: 91:1-91:15.

9. Small Stretch Pairwise Spanners and Approximate D-preservers.
   Telikepalli Kavitha and Nithin Varma.
   SIAM *Journal of Discrete Mathematics* 29(4): 2239-2254 (2015).

10. Small Stretch Pairwise Spanners.
    Telikepalli Kavitha, Nithin Varma.
    Proceedings of ICALP (1) 2013: 601-612.

11. Rainbow connection number and connected dominating sets.
    L. Sunil Chandran, Anita Das, Deepak Rajendraprasad and Nithin Varma.
    Journal of Graph Theory 71(2): 206-218 (2012).

12. Rainbow Connection Number and Connected Dominating Sets.
    L. Sunil Chandran, Anita Das, Deepak Rajendraprasad and Nithin Varma.
    Electronic Notes in Discrete Mathematics 38: 239-244 (2011).