Sublinear Distributed Reconstruction

Seshadhri Comandur

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

Advised by
Bernard Chazelle

November 2008

Abstract

Online property reconstruction: Large data sets are ubiquitous today with the advent of high speed computing and the explosion of data storage capabilities. Using this data as an input is quite a challenge, since we do not even have the luxury of reading the whole data. Linear time algorithms are too slow, and we are forced to move into the realm of *sublinear time* algorithms.

Usually, data sets are useful because they satisfy some structural property. An ordered list of numbers may be in sorted order, and this property may be central to the usefulness of this list. Such a property is very sensitive to noise. Given a large data set f that does not satisfy some such property $\mathcal{P}$, can we modify f minimally to make it have $\mathcal{P}$? Furthermore, this data set is large and we may not have access to it all at once. We would ideally like to make this change *online*.

These considerations led to the formulation of the *online property reconstruction* model, which we investigate in this thesis. We are given oracle access to a function f (defined over an appropriate discrete domain) which does not satisfy a property $\mathcal{P}$. We would like to output a function g which satisfies f and differs from f at very few values. The function g is output by a sublinear time online procedure, called a *filter*. Such a procedure takes as input a domain point x, and outputs $g(x)$ in time sublinear to the domain size. We design filters for the following scenarios:

1. *Monotonicity* : The functions are real-valued on the d-dimensional domain $[1, n]^d$, and the property to be maintained is monotonicity. This is a natural starting point for discussing the online reconstruction model, and involves the development of many sublinear techniques to study monotonicity. We show the surprising result that once a random seed of size $(d \log n)^{O(1)}$ is fixed, the value of $g(x)$ (for any input domain point x) can be computed in $(\log n)^{O(d)}$ time. These results are provided in Chapter 2 and are joint work with Michael Saks.

2. *Convexity* : We take property reconstruction to the geometric world and study convexity in two and three dimensions. Given a polygonal chain or a 3D terrain, we design filters that minimally modify the input and make it convex. Especially for 3D, we require a large set of geometric tools. We also prove lower bounds showing a complexity gap between 2D and 3D. This is joint work with Bernard Chazelle and given in Chapter 3.

3. *Expansion* : The input is a bounded degree graph which we want to make into an expander. The main algorithmic technique used here is random walks. We initiate a discussion into the behavior of random walks in graph that are almost expanders - graphs formed by arbitrarily connecting a small unknown graph to a large expander. We show interesting mixing properties of walks in such graphs, and use this to construct a filter for expansion. These results are given in Chapter 4 and are joint work with Satyen Kale and Yuval Peres.

Self-improving algorithms: A real world algorithm performs the same task, say sorting, repeatedly on inputs coming from some source. It is natural to assume

that this source, although unknown, may possess some structure that allows for faster running time. We investigate ways in which an algorithm can improve its expected performance by fine-tuning itself automatically with respect to an *arbitrary, unknown* input distribution. Assume that the inputs to the algorithm are generated indepedently from a fixed distribution $\mathcal{D}$. A *self-improving* algorithm learns about $\mathcal{D}$ and then makes optimizations specific to $\mathcal{D}$. This allows it to beat the worst-case running time for the given problem. We give such self-improving algorithms for sorting and computing Delaunay triangulations. The highlights of this work: (i) an algorithm to sort a list of numbers with optimal expected limiting complexity; and (ii) an algorithm to compute the Delaunay triangulation of a set of points with optimal expected limiting complexity. These results are described in Chapter 5 and are joint work with Nir Ailon, Bernard Chazelle, Ken Clarkson, Ding Liu, and Wolfgang Mulzer. In both cases, the algorithm begins with a training phase during which it adjusts itself to the input distribution, followed by a stationary regime in which the algorithm converges to its optimized incarnation.

Acknowledgments

This thesis may have only my name on the title page, but my own contribution and effort are just a few ingredients of this work. My advisor Bernard Chazelle deserves to get a lot more than simply a mention in this section. When I joined graduate school five years ago, I was very inexperienced and naïve in the ways of research. Bernard took me under his wing and slowly led me to approachable problems. I found it rather hard initially to work on open-ended research problems, and it was Bernard's efforts and confidence in me that helped me get over these difficulties. Once I started getting more confident and maturing, he made me move towards becoming an independent thinker. Finally, in the last years at Princeton, he gave me the complete freedom I required to pursue my own ideas and agendas. It was his brilliant ideas and sound advice that have made me the researcher I am today, and I am ever grateful for that.

I would also like to acknowledge the various grants that funded my research (and me) : NSF grants CCF-0634958, CCR-998817, CCR-0306283, and ARO Grant DAAH04-96-1-0181.

The theory group at Princeton is, in a word, amazing, and it was a pleasure to spend five years there. A significant part of what I learned came from the discussions with and courses taught by various faculty members : Sanjeev Arora, Boaz Barak, Moses Charikar, Rob Schapire, Bob Tarjan, and others. It was great to be around lots of smart and friendly students with whom I could share my half-baked (and usually bad) ideas. To name a few : Amit Agarwal, Nir Ailon, Eden Chlamtac, Moritz Hardt, Elad Hazan, Satyen Kale, Ding Liu, Neeraj Kayal, Konstantin and Yury Makarychev, Wolfgang Mulzer, Nitin Saxena, among others. Special thanks to Eddie and Wolfi. I don't know how I could have passed the difficult and dull times without them. They were there to listen to my ideas, my thoughts, my complaints, and my problems.

This thesis is a collection of results that were obtained with many others : Nir Ailon, Bernard Chazelle, Ken Clarkson, Elad Hazan, Ding Liu, Satyen Kale, Wolfgang Mulzer, Yuval Peres, and Mike Saks. To all my co-authors, thank you. This thesis is as much about their work as it is about mine. Their intuition and ideas are all over the following pages. A few special mentions are in order here. Thanks to Ken and Elad for the wonderfully productive summer at IBM Almaden. I would like to especially thank those who made my summer internship there possible. Thanks a lot of Satyen for the many invitations to Microsoft Research, Redmond. Last but certainly not least, I am very grateful to Mike Saks for our collaboration together (and for the many meetings and coffees in New Brunswick). It was a real pleasure to work with a stalwart like him. Our work is special to me because it was my first research project done outside of Princeton. It was, in some sense, my first step towards becoming an independent researcher, and it couldn't have been possible without Mike's comments and penetrating insights.

Five years can feel like a long time, or a very short time, depending on how much fun one has. For me, it has been the latter, and a lot of credit for that goes to the 3K-3M-7H crowd : Sunil Ahuja, Gaurang Bhargava, Manish Dubey, Amit

Kumar, Rohit Rajgaria, Arun Raman, Easwaran Raman, Prateek Sharma, and Varadharajan Srinivasan. Thanks for the wonderful times at Princeton, the trips outside, the squash and Ultimate games, the freshly cooked food, and all the fish. I wouldn't have survived without that. To the Arsha Bodha Center and Swamiji, I doubt that I can express my gratitude in words.

None of this would have been even remotely possible without the support and love of my family. To my parents, C. Venkatesan and Ramya Venkatesan, and my sister Roopa, I humbly dedicate this thesis.

To Divya, thanks. Sometimes less words say more.

To my parents, and my sister

Contents

List of Figures

Chapter 1

Introduction

The first step that a traditional algorithm usually takes is to read the whole input. In the past, algorithms were always allowed time at least linear in their input size. This view of algorithms has changed considerably in the past decade. From a theoretical perspective, algorithms that did *not* read their input completely were used in proving the PCP theorem [17, 18]. Concepts such as self-testing and self-correction [23] required algorithms that gave some verification of program correctness *without* checking all possible outputs. In the real world, the size of data sets has been increasing at an exponential rate, and reading the whole input itself is an unaffordable luxury. We need *sublinear time* algorithms that read a vanishingly small fraction of their input and still say something intelligent and non-trivial about input properties.

The model of *property testing* [72] has been very useful in understanding the power of sublinear time. In the standard algorithmic setting, we accept an input when it satisfies some property and reject it otherwise. We relax this condition by introducing the notion of *distance* to a property. The algorithm, called a property tester, accepts inputs that satisfy the property and reject those that are *far* from satisfying it. All other inputs can be dealt with arbitrarily. Sublinear, or even constant time algorithms are surprisingly good at this task. In many circumstances, they can even approximate certain parameters of the input. In *local* and *list decoding* [16, 78], we have algorithms that again infer properties from noisy codewords by reading a very small portion of them.

Philosophically, the major question is : what can be known without reading the whole input? One of the aims of this thesis is to push the boundaries of what sublinear time can achieve. We introduce the model of *sublinear distributed reconstruction* and show many sublinear time algorithms in this model. These algorithms are far more powerful than property testers and allow us to modify an input to give it some structural property (instead of only telling us whether the input possesses it or not).

In this vein of attempting to go beyond "standard lower bounds" (i.e., the linear time lower bound for processing an input), we introduce *self-improving algorithms*. The previous topic was about designing sublinear time algorithms to deal with massive inputs. In a different direction, we build algorithms that exploit input *dis-*

tributions and optimize running times. Normally, an algorithm is created to process a fixed task repeatedly (on different inputs). Indeed, in such a situation, it is highly likely that the input is generated by some unknown and unrepresentable distribution. Can we somehow glean enough information about this distribution and improve our running time? In this thesis, we construct such self-improving algorithms that have the ability to adapt themselves to the current input distribution.

1.1 Online property reconstruction

The process of assembling large data sets is prone to varied sources of error, such as measurement error, replication error, and communication noise. Error correction techniques (i.e. coding) can be used to reduce or eliminate the affects of some sources of error, but some residual errors may be unavoidable. Despite the presence of such inherent error, the data set may still be very useful.

One problem in using such a data set is that even small amounts of error can significantly change the behavior of algorithms that act on the data. For example, if we do a binary search on an array that is supposed to be sorted, a few erroneous entries may lead to behavior that deviates significantly from the "correct" behavior.

This is an example of a more general situation. We have a data set that ideally should have some specified structural property, i.e., a list of numbers that should be sorted, a set of points that should be in convex position, or a graph that should be a tree. Algorithms that run on the data set may rely on this property. A small amount of error may destroy the property, and result in the algorithm producing wildly unexpected results, or even crashing. In these situations, a small amount of error may be tolerable only if the structural property is maintained.

The extensive theory of property testing algorithms (see surveys [43, 45, 70]) provides means to detect such problems in existing data sets in sublinear time. However, in applications it may not suffice to just detect such problems, since one needs to actively process the data to perform useful tasks. Usually, it is reasonable to minimally modify the data to restore the property, so that further processing can take place without problems. Yet, since the input is so massive, one cannot afford to read the whole input and then fix it.

Thus, this may seem like an impossible task, but usually access to the large input is provided in the form of local queries: for example, if the input is, say, the web graph, then typical queries to the input would ask for all the outgoing links from a given webpage. In such cases, we would like to repair the input as and when we read it in *sublinear time*.

These considerations motivate the formulation of the *online property reconstruction* model (introduced in [6]). We are given a data set, that we think of as a function f defined on some domain Γ. This can be used to model geometric data as well as graphs. Ideally, f should have a specified structural property $\mathcal{P}$, but this property may not hold due to unavoidable errors. We wish to construct *online* a new data set g such that (1) g has property $\mathcal{P}$ and (2) $d(g, f)$ is small, where $d(g, f)$ is the fraction of values $x \in \Gamma$ for which $g(x) \neq f(x)$.

How small should $d(g, f)$ be in condition (2)? Define $\varepsilon_f = \varepsilon_f(\mathcal{P})$ to be the minimum of $d(h, f)$ over all h that satisfy $\mathcal{P}$. Of course, ε_f is a lower bound on the deviation of g from f. The *error blow-up* of g is the ratio $d(g, f)/\varepsilon_f$. This error blow-up can be viewed as the price that is paid in order to restore the property $\mathcal{P}$, and we want this to be a small constant.

An offline reconstruction algorithm explicitly outputs such a g on input f. In the context of large data sets, the explicit construction of g from f requires a considerable amount of computational overhead (at least linear in the size of the data set). Instead, we want to have an algorithm, called a *filter*, that given a domain point $x \in \Gamma$ outputs $g(x)$ in *sublinear* ($o(|\Gamma|)$) time.

1.1.1 Distributed property reconstruction

We now consider a more general notion of reconstruction than the one above. This new notion will generalize property testing (this shall be explained in more detail in Section 1.1.2). We motivate this by describing a distributed setting for reconstruction. There are many independent users who want to use the large reconstructed data set g. The function g was output using a filter, which is necessarily a randomized algorithm (each query must be handled in sublinear time, and that certainly needs randomness). Since we wish to provide all users that *same* function g, we would like to construct a *distributed filter*. Consider a small random sublinear seed ρ. Any user who wants access to g is provided with ρ. The distributed filter is a sublinear time algorithm that only uses ρ for random bits[1]. Given domain point $x \in \Gamma$, the filter outputs $g(x)$. Note that because all users have the same random seed, they all see the same function g. Since the random seed is sublinear and the running time of the filter is sublinear, the total space that is ever needed is sublinear.

We now provide a formal definition. A *distributed filter* [74] for reconstructing property $\mathcal{P}$ is an algorithm A that has oracle access to a function f on domain Γ and to an auxiliary random string ρ (the "random seed"), and takes as input $x \in \Gamma$. For fixed f and ρ, A runs deterministically on input x to product an output $A_{f,\rho}(x)$. We want A to satisfy the following properties:

1. For each f, with high probability (with respect to the choice of ρ), the function $A_{f,\rho}$ should satisfy $\mathcal{P}$.
2. For each f, with high probability (with respect to the choice of ρ), the function $A_{f,\rho}$ should be "suitably close" to f.
3. For each x, $A_{f,\rho}$ on x can be computed very quickly.
4. ρ should be "much smaller" than $|\Gamma|$.

Since the filter is essentially a randomized procedure, the guarantees of the properties of the function $A_{f,\rho}$ are probabilistic (we usually require that these hold with with very high probability over the random seed ρ). For a distributed filter,

[1] Alternatively, the filter is a deterministic procedure that takes ρ as input.

there are three parameters of interest: the error blow-up, the time per query and the number of random bits needed for ρ, that is, to initialize the filter. The memory used by a distributed filter is bounded by the sum of the length of ρ and the maximum time of a single query.

As we mentioned before, if ε_f is the distance of f to the property $\mathcal{P}$, we want $d(f, g) = O(\varepsilon_f)$. The running time and the size of ρ should be bounded by $o(|\Gamma|)$. Ideally, we would like this running time to be *independent* of ε_f. In other words, the amount of time we spend for reconstruction does not depend on how far f is from satisfying $\mathcal{P}$. Interestingly, we will show that such a filter is not possible for reconstructing convexity in 3-dimensions.

A much weaker notion of reconstruction is that of *sequential reconstruction* [6], which actually predates its distributed cousin. Here, we assume that the domain point queries are coming in sequential order $x_1, x_2, \cdots$. Given query x_i, we wish to output $g(x_i)$ in sublinear time. We are allowed to store all the previous answers, and thereby, actually use linear space in the long run. Furthermore, the function g may depend on the sequence of input queries.

Although distributed reconstruction is a more natural (and possibly cleaner) concept, thinking about sequential reconstruction seems to be a very helpful step in designing distributed filters. Indeed, for the problem of monotonicity reconstruction, the results on sequential filters [6] provided the necessary insight to build distributed ones. Also, sequential filters can usually be made to run faster and have lower error blow-up.

An interesting side-effect that we get out of studying properties from the perspective of reconstruction is the variety of tools developed. The present array of tools for sublinear algorithms and property testing are usually inadequate for reconstruction, and we are forced to delve deeper into the problem. As a result, we end up with newer and more powerful techniques for sublinear algorithms, and a list of questions of independent interest.

In this thesis, we will present filters for monotonicity, convexity, and for expansion. We will not present the results on sequential filters for any of these problems. The description of the distributed filters are self-contained and no further insight is provided their simpler sequential forms.

1.1.2 Previous work and relation to property testing

One case of property reconstruction that has been extensively studied is *error correcting codes*. If $C \subseteq \{0,1\}^n$ is such a code and $\mathcal{P}$ is the property of being a code word then the error correction problem for C is the same as the reconstruction problem for property $\mathcal{P}$. Distributed property reconstruction corresponds to *local decoding* of the code. Note that in general, reconstruction can be much harder than error correction - in error correction, we assume that there is only one codeword (or a few words, in list decoding) that are "close" to the given word. On the other hand, for a general property $\mathcal{P}$, there may be many functions that are in $\mathcal{P}$ and are sufficiently close to the given function. This makes reconstruction a lot harder, because there is no unique correct output for the filter.

There is a similar connection to the concept of program checking and self-correction, as introduced by [23]. Here, we can think of the input function as a somewhat buggy program that is supposed to perform a fixed task, and we wish to correct the faulty outputs. Again, there is only one possible correct output function. There has been a lot of work regarding list-decoding of low-degree polynomials [16, 78] (where the property we wish to enforce is that the input is a set of values generated by a low-degree polynomial), that can be seen as a special case of property reconstruction.

A slightly related notion of reconstruction was discussed in [47], for generalized partition problems in dense graphs. Given a input dense graph that satisfies some partition property (say k-colorability), we wish to construct a partition of the vertices that has an ε-fraction of violating edges. The reconstruction here can be done with a small random subset (in this case, some function of ε independent of n) of the vertices. For any other vertex, we can decide the set of the partition it belongs to by looking at only the small random subset. This is very reminiscent of a distributed filter, since each decision can be made in sublinear time [2].

In [71], a sublinear time algorithm to approximate the vertex cover of a graph is constructed from a distributed algorithm. This shows an interesting connection between distributed and sublinear algorithms. Distributed filters can be seen as going in the opposite direction : we use sublinear tools to get a distributed algorithm.

In property testing [72], we are given a property $\mathcal{P}$ and a distance parameter $\varepsilon \in [0, 1]$. Given a function f, we wish to distinguish between the following cases : (1) f has the property $\mathcal{P}$, and (2) f is ε-far from satisfying $\mathcal{P}$,i.e., $\min_{g \in \mathcal{P}} d(f, g) > \varepsilon$. There has been a large amount of work done on property testing in various models (see surveys [43, 45, 70]). The problem of monotonicity in the context of property testing has been studied in [39,41,46]. Geometric testing problems have been looked at in [32,36]. More closely related to reconstruction is the notion of tolerant property testing [68] which deals with actually estimating the distance in sublinear time. Sublinear algorithms for determining the distance of a function to monotonicity have been given in [5, 68].

We note that distributed reconstruction is a strict generalization of these settings. Given a distributed filter for a function f, we can query it at a random domain point x and check if $g(x) \neq f(x)$. By repeating this test suitably many times, we can actually estimate ε_f (upto a multiplicative factor of the error blow-up) in sublinear time.

1.1.3 Monotonicity reconstruction

We start the discussion of distributed reconstruction with the property of monotonicity. The most basic case of this problem is to consider the universe of functions of the form $f : [n] \to \mathbb{R}$ with the property $\mathcal{P}$ of all monotonically non-decreasing functions (think of sorted arrays). The most natural extension into higher dimen-

[2] We add that the running time, in these algorithms, depends on the parameter ε, whereas our filters have running time independent of that.

sions is to consider functions of the d-dimensional lattice - $f : [n]^d \to \mathbb{R}$. Instead of a total order on the domain, we have a partial ordering. A domain point x is less than y if all coordinates of x are less than the respective coordinates of y. Such a function f is monotone if, for any x less than y, $f(x) \leq f(y)$.

In Chapter 2, we construct a distributed filter for monotonicity for these functions with the following performance:

- The time per query is $(\log n)^{O(d)}$.
- The error blow-up is $2^{O(d^2)}$, independent of n.
- The number of random bits needed to initialize the filter is $(d \log n)^{O(1)}$.

The running time per query is strongly sublinear in the domain size, and at some level, it is actually surprising that such a filter can be constructed. We give more details on the technical difficulties later on. The basic ideas and techniques for this filter come from sublinear time algorithms for estimating the distance to monotonicity [5]. We greatly extend and enhance these tools. One of the interesting features of this result is the construction of a data structure that allows us to sample, in sublinear time, contiguous portions of the domain at various scales.

1.1.4 Convexity reconstruction

In the geometric realm, we consider the problem of convexity reconstruction. In the two-dimensional case, the input is a polygonal chain represented by a doubly-linked list. In three dimensions, we are provided with a terrain : this is a triangulated planar graph in the xy plane with z-coordinates assigned to every vertex. By linear interpolation, we get a triangulated three-dimensional surface. The aim is to generate a convex surface that is close to the input, using a distributed filter. The filter modifies the coordinates of the vertices to make the terrain convex. The queries are made for the vertices of a face (or an edge, in two-dimensions).

In the following, n denotes the size of the input and the distance between two terrains (polygons) is defined as the minimum number of faces (edges) whose coordinates need to be modified to transform one into the other. In Chapter 3, we give filters for reconstructing convexity in two and three dimensions:

1. A $\widetilde{O}(n^{2/3})$ time[3] distributed filter for reconstructing the convexity of a simple polygon represented as a doubly-linked list. We assume that each vertex is labeled with its rank in the list. To show a gap between property testing and reconstruction, we also give an almost optimal $\widetilde{O}(n^{1/3})$ time property tester and prove an $\Omega(n^{1/2})$ lower bound for the running time of a filter.
2. A distributed filter for reconstructing the convexity of a bounded aspect ratio[4] terrain presented in triangulated DCEL format with a worst case query time

[3]The notation $\widetilde{O}(\cdot)$ hides polylogarithmic factors in n.

[4]A terrain is said to have *bounded aspect ratio* if the xy-projections of the faces have bounded sides and angles.

of $O(n^{12/13+\delta} + \varepsilon_{\mathcal{D}}^{-O(1)})$. In the DCEL (doubly-connected edge list, look at Section 2.2 in [37]) format, we have vertex, edge, and face tables. For each vertex, we have access to the real coordinates of the corresponding point in space, and to all incident edges in cyclic order. For each edge, we have pointers to the incident faces, and for each face, we have access to bounding edges in cyclic order. This data structure allows us to perform walks on the terrain, and obtain geometric information in this process. Here, δ is an arbitrarily small positive constant and $\varepsilon_{\mathcal{D}} n$ is the terrain's distance to convexity, ie, the minimum number of faces whose coordinates need to be modified in order to make the terrain convex. We also prove a lower bound that explains why the complexity must depend on $\varepsilon_{\mathcal{D}}$.

The error blow-up in both cases is $O(1)$.

Many tools are required to achieve this result, including the planar separator theorem, balanced trapezoidal decompositions, and sublinear time approximation algorithms for vertex cover. A completely new technique that we apply here is sampling in range spaces of *unbounded* VC dimension. Another puzzling fact is that the 2D filter's complexity does *not* depend on the distance $\varepsilon_{\mathcal{D}}$ but its 3D counterpart does. By showing that online 3D reconstruction requires $\Omega(\varepsilon_{\mathcal{D}}^{-1})$ time, we prove that this difference is intrinsic and represents yet another complexity gap between 2D and 3D. Of independent interest, we also give a sublinear time algorithm that computes small balanced separators for geometrically embedded planar graphs. It is unlikely that this construction is optimal and we pose an intriguing problem : the construction of optimal sized separators in sublinear time.

In related previous work, offline geometric reconstruction has been studied before, but usually the metric is geometric (like the Hausdorff distance) and not combinatorial. Examples include finding the best approximation of surfaces satisfying certain criteria [1–3, 12]. On the other hand, a notion of combinatorial distance is certainly present when studying the computational aspects of the Erdős-Szekeres theorem [29] or other Ramsey-like results. Geometric properties have been well studied within the purview of property testing [32, 36], program checking [63], and sublinear algorithms [28]. Efficient testers have been given for convexity [32, 36, 41], clustering [10, 53, 54, 65], and Euclidean MST [31, 33], but there too the relevance to our work is only tangential.

1.1.5 Expander reconstruction

The input is a huge bounded degree graph represented by adjacency lists. We wish to make the graph into an *expander*, i.e. for any subset of vertices of at most half the size of the graph, the number of outgoing edges (i.e., to the complement of the subset) is a significant fraction of the total number of edges incident on the vertices in the set. The queries are in the form of requests for the adjacency list of a specified vertex.

More formally, we are given a bounded degree graph $G = (V, E)$, with n vertices, represented by adjacency lists. All vertices have degrees bounded by some fixed constant d. We are also provided a conductance parameter $0 < \phi < 1$ - we wish to make the conductance of G as close to ϕ as possible, while changing the graph minimally (and maintaining the degree bound). The parameter ϕ should be thought of as a constant. In Chapter 4, we design a distributed filter with a running time of $\widetilde{O}(\sqrt{n}/\phi^2)$ per query[5] that outputs a graph G' of conductance $\phi^2/\log n$. The symmetric difference of G and G' is $O(\frac{1}{\phi}\text{OPT})$, where OPT is the optimal number of edges (of G) to be changed to make the conductance at least ϕ. Note that for constant ϕ, the filter changes only a constant factor more than the optimal change necessary.

We show that expander reconstruction is connected to some intriguing questions about random walks in "noisy" expanders. Suppose one is given a graph G that consists of a large expander connected arbitrarily to some small unknown graph (the noise). The graph G is probably not an expander, but we would like to show that random walks on the expander part are not greatly affected by the noise. Turning this around, we can ask: what portion of the large expander is affected by this noise? Can we still hope that from almost all of the expander, we can reach a vast majority of the expander by short random walks? We prove the rather strong statement that the noise can only affect a part of the expander that is proportional in size to the noise. We believe that this result should be useful in other contexts as well, and we present it in the general setting of arbitrary irreducible Markov chains rather than for our specific application.

For the reconstruction problem, we design a sublinear time procedure based on random walks that can very accurately distinguish between vertices in a high expansion part of the graph, and vertices of the noisy part. This procedure essentially measures the distance between the probability distribution induced by a random walk on an undirected graph to the stationary distribution. Previous algorithms for measuring such distances [20] are not efficient enough for our situation, and we design new tools for these scenarios. Using our general result on the noise-tolerance of expanders, we show that the number of vertices from which a random walk mixes poorly is a good measure of the combinatorial distance (in terms of edges changed) of a graph to being an expander. This establishes a link between these two different notions of distance.

1.2 Self-improving algorithms

The classical approach to analyzing algorithms draws a familiar litany of complaints: worst-case bounds are too pessimistic in practice, while average-case complexity too often rests on unrealistic assumptions. Efforts have been made to analyze algorithms under more complex models (e.g., Gaussian mixtures, Markov model outputs) but with limited success and lingering doubts about the choice of priors.

[5]The size of the random seed is also $\widetilde{O}(\sqrt{n}/\phi^2)$.

Ideally, one would like to compute a function f with the help of a *self-improving* algorithm. Upon receiving its first input instance I_0, such an algorithm would compute $f(I_0)$ with, say, good worst-case guarantees and nothing more. Think of newly installed software that knows nothing about the user and runs in its "vanilla" configuration. Subsequently, as it is called upon to compute $f(I_k)$ for $k = 1, 2, \ldots$, the algorithm would gradually improve its performance through automatic finetuning. Intuitively, if the I_k's are drawn from a low-entropy distribution, the algorithm should be able to spot that and *learn* to be more efficient.

The obvious analogy is data compression, which seeks to exploit low entropy to minimize encoding size. Our aim is : given an *unknown* distribution $\mathcal{D}$, design a self-improving algorithm whose running time that converges to the optimal expected running time. The second goal, which is to optimize the convergence speed, is more of a machine learning nature. One of the surprises of this work is how minimal distribution learning suffices for dramatic self-improvement.

The starting point of this research is the observation that, trimmed of noise, real-world data is often of much lower entropy than size alone suggests. Hidden Markov models for speech recognition can be remarkably effective with only a few thousand states. Anecdotal evidence can also be gleaned from the current trend toward personalization in the design of web tools (search engines, recommendation systems, etc). Input data is often lodged in a tiny slice of input space that cannot be captured by closed-form distributions. To make predictions about the slice is the essence of machine learning [40, 56, 66]. To take computational advantage of the slice is what self-improving algorithms are all about.

1.2.1 Results

The performance of a self-improving algorithm is measured with respect to an *unknown* memoryless random source $\mathcal{D}$ of input instances. The algorithm is given instances $I_0, I_1, \ldots$ drawn independently from $\mathcal{D}$, which it must solve one at a time in batch mode with: (1) no prior knowledge of future instances, that is, $f(I_k)$ must be computed before any of the I_j's ($j > k$) are known; and (2) no prior knowledge of $\mathcal{D}$. The algorithm may store auxiliary information to help improve its performance. (Unlike self-organizing data structures, however, none of that information should be *necessary* for the algorithm to complete its task.) We use $\mathcal{D}$ as shorthand for $\mathcal{D}^n$, the n-th member of an infinite ensemble of distributions—one for each input size. After a training phase, we expect the algorithm to settle into its steady state whose expected running time is called its limiting complexity. Note that from the user's perspective the only difference noticeable in the training phase is that the system might be a little slower.

Our first result is, in some sense, the first truly optimal sorter. Given a source $\mathcal{D}$ of real-number sequences $(x_1, \ldots, x_n)$, let $\mathcal{D}_<$ be the distribution over the symmetric group induced by the ranks of the x_i's (using the indices i to break ties). The complexity of our algorithm depends on the entropy $H(\mathcal{D}_<)$. Note this quantity can be much smaller than the entropy of the source itself but can never exceed it.

- SORTING: We give a self-improving algorithm with a limiting complexity of $O(H(\mathcal{D}_<) + n)$ and prove that it is optimal. If the input $(x_1, \ldots, x_n)$ to be sorted is obtained by drawing each x_i independently (from a distribution that might depend on i), then for any $\varepsilon > 0$ the storage can be made $O(n^{1+\varepsilon})$ for an expected running time of $O(\varepsilon^{-1}H(\mathcal{D}_<) + n)$: this tradeoff is optimal for distributions of high enough entropy. The training takes $O(n^\varepsilon \log n)$ rounds. We also show that independence is necessary: without it, the storage *must* be exponential in n.

We take the concept of self-improving algorithms to the geometric realm and address the classical problem of computing the Delaunay triangulation of a set of points in the Euclidean plane.

- DELAUNAY TRIANGULATIONS: Assuming a distribution of n points, each one drawn independently from its own unknown (arbitrary) random source, we give a self-improving algorithm of optimal limiting complexity. We get time-space tradeoffs as well as lower bounds similar to those for sorting.

1.2.2 Previous work

Related concepts have been studied before. List accessing algorithms and splay trees are textbook examples of how simple update rules can speed up searching with respect to an adversarial request sequence [7, 24, 52, 76, 77]. It is interesting to note that self-organizing data structures were investigated over stochastic input models first [9, 22, 51, 61, 69, 73].

Algorithmic self-improvement differs from past work on self-organizing data structures and online computation in two fundamental ways: (i) self-improving algorithms operate offline and do not lend themselves to competitive analysis; (ii) they do not exploit structure within any given input but, rather, within the ensemble of input distributions. For example, suppose that the distribution $\mathcal{D}$ consists of two random but *fixed* permutations, each one equally likely. Any solution in the adaptive, self-organizing/adjusting framework requires $\Omega(n \log n)$ time. It is trivial, however, to design a self-improving algorithm of linear limiting complexity: sort the two permutations and store them; given any input instance, apply both permutations separately and output the permuted instance that is sorted.

Chapter 2

Monotonicity Reconstruction

2.1 Introduction

Our first result on distributed reconstruction is that of monotonicity reconstruction. The input is a function $f : [1, n]^d \to \mathbb{R}$, and we wish to output a monotonically non-decreasing (with respect to a partial order on the domain) function g. The function g is generated by a sublinear time distributed filter.

Monotonicity has been well-studied, from a property testing perspective [39, 41, 46]. The testing techniques have been used to devise approximation algorithms [5,68] for the distance to monotonicity. The present result can be viewed as the next step in this sequence of work. The filter we design goes beyond approximating the distance to monotonicity and tells us (at any domain point) what the function value (of a close enough monotone function) should be.

2.1.1 Preliminaries

We consider functions defined on a fixed finite domain Γ. A property $\mathcal{P}$ is a set of functions defined on Γ. The distance between two functions f and g, denoted $d(f, g)$ is the fraction of $x \in \Gamma$ for which $f(x) \neq g(x)$. For a function f and a property $\mathcal{P}$, the distance of f to $\mathcal{P}$, $\varepsilon_f = \varepsilon_f(\mathcal{P})$ is the minimum of $d(f, h)$ for $h \in \mathcal{P}$.

For a positive integer m, $[m]$ denotes the set $\{1, 2, \ldots, m\}$. Throughout this paper, $\Gamma = [n]^d = \{(x_1, \ldots, x_d) : \forall i \in [d], x_i \in [n]\}$ for some integers n and d. We fix n and d, and assume, without (much) loss of generality that $n = 2^k$ where k is a positive integer. Γ is partially ordered with respect to the product relation: $x \leq y$ if and only if $x_i \leq y_i$ for all $i \in [d]$. Elements of Γ are called *points*. Points are generally denoted by lower case letters, sets of points are denoted by upper case letters and sets of sets of points by caligraphic letters.

We consider functions mapping Γ to the nonnegative reals (for simplicity of presentation). Such a function f is *monotone* if $f(x) \leq f(y)$ whenever $x \leq y$. Note that the range is not discrete and has no upper bound. We do not use the actual value of the function in any non-trivial way (being more general than many monotonicity testers that use a bounded and discrete range [39, 46]).

As defined in the introduction, a distributed filter uses randomness only in the choice of the string ρ that initializes the filter. All probability statements are made with respect to the choice of this string ρ. In general, when we say that an event occurs with *low probability* we mean that its probability is $1/|\Gamma|^{\omega(1)}$, i.e. superpolynomially small in $|\Gamma|$. Conversely, a high probability event is one having probability $1 - (1/|\Gamma|^{\omega(1)}))$.

The main theorem of this chapter is :

Theorem 2.1.1. *Let ρ be a random seed of length $(d \log n)^{O(1)}$. There exists a determinstic process $A_{f,\rho}$ (the distributed filter) with the following properties that takes as input a domain point x and outputs a real number.*

- *With high probability, the function g defined by $g(x) := A_{f,\rho}(x)$ is monotone.*
- *With high probability, the function g differs from f at at most $2^{O(d^2)}\varepsilon_f n^d$ points.*
- *The running time of $A_{f,\rho}(x)$ is $(\log n)^{O(d)}$.*

2.1.2 Why are distributed filters hard to design?

The starting point for the construction of our distributed filter for monotonicity is the online sequential filter of [6], which in turn is based on sublinear time approximations for ε_f [5]. We now give the main ideas of their construction, and indicate the difficulties in making their construction distributed. In the discussion below, when we say an algorithm is "fast", we mean that it runs in time polylogarithmic in $|\Gamma|$.

We start with the case $d = 1$, i.e., the one-dimensional case. The basic idea (implicitly used) in [6] is to classify the domain points as *good* and *bad* in such a way that the following conditions hold -
(1) There is a fast algorithm for testing whether a given point is good or bad.
(2) There are not many bad points.
(3) The function restricted to the set of good points is monotone.

The third property ensures that it is possible (though not necessarily efficiently possible) to change the function only on bad points and make the function monotone. To do this define $m(x)$ for $x \in \Gamma$, to be the largest good point less than or equal to x, and define $g(x) = f(m(x))$. It is easy to see that this yields a monotone function.

Note that a filter that can do such a classification in sublinear time can immediately be used for *approximating the distance to monotonicity*. As mentioned in the introduction, reconstruction is a harder problem than property testing or tolerant testing.

In [6], a point x is classified as bad if (roughly) there is an interval around x that contains a large fraction of points whose f values are out of order with respect to $f(x)$. With this good/bad classification there seems to be no fast way to compute $m(x)$. Instead, given query point x, the filter in [6] finds a good point y that is a "sufficiently close" random approximation to $m(x)$ and sets $g(x)$ to be $f(y)$.

Choosing a random approximation to $m(x)$ rather than $m(x)$ creates a significant problem: we are no longer guaranteed that the function g defined in this way is monotone. For example, suppose $y < m(x)$ is the approximation to $m(x)$, and $f(y) < f(m(x))$. Suppose that after setting $g(x)$ to $f(y)$, a query is made to reindex $m(x)$. Since $m(x)$ is good, we want to define $g(m(x)) = f(m(x))$, but this will violate monotonicity with the already defined $g(x) = f(y)$.

In online reconstruction, this problem can be handled since the algorithm can save the previously answered queries in a sorted list and impose the condition that future g values be consistent with previously assigned g values. This is what is done in [6].

Distributed reconstruction does not have this luxury. Thus the main challenge in designing a distributed filter is to redefine the notion of good and bad in a way that will allow us to quickly find $m(x)$ exactly for any given x. Why is this difficult? Since $m(x)$ may be quite far away from x, and we want to find $m(x)$ quickly it would seem that the algorithm would need to use random sampling to explore the vicinity of x. Such sampling will provide only an approximate picture of the vicinity of x, but the closest good point is something *exact* and we cannot tolerate any error in determining it. Our new definition of good/bad is a somewhat involved modification of the definition in [6] that is carefully designed to allow us to compute $m(x)$ quickly and exactly. A priori, it is unclear that such a redefinition is even possible. The notion of "good" and "bad" points does not appear explicitly in the description of our algorithm, though it provides a useful intuition for what is going on.

The difficulties in designing a distributed filter are substantially greater for the case of higher dimensional domains ($d \geq 2$). Suppose we had a definition of good and bad satisfying the three conditions stated in the one-dimensional case. It is still true that, in principle, it is possible to define a monotone g that agrees with f on all good points. But explicitly computing such a g is more complicated. Given x, let $M(x)$ be the set of points which are maximal in the set of good points less than or equal to x. In the one-dimensional case, $M(x)$ has one element $m(x)$, but in the multi-dimensional case, where the domain is not totally ordered, this is not the case. Still if we define $g(x)$ to be the maximum of $f(y)$ for $y \in M(x)$, then the resulting g is monotone. To implement this, one would have to find all of the elements of $M(x)$. Even when $M(x)$ has size 1 (as in the one-dimensional case) this is difficult, but here the difficulty is compounded because $M(x)$ might be as large as $\Omega(n^{d-1})$, and we need our computation to run in time polylogarithmic in n. In [6] this is done by finding a polylogarithmic size set $Rep(x)$ which is a suitable approximation to $M(x)$, and then defining $g(x)$ to be the maximum of $f(y)$ for $y \in Rep(x)$. As with the one-dimensional case, using an approximation to $M(x)$ destroys the guarantee that the g defined in this way is monotone, so one must save the values of g to all queries, and impose the additional requirement that queries are mutually consistent.

Since a distributed filter cannot afford to do this kind of approximation, one approach is to find a definition of good/bad that satisfies the three conditions above and one more besides: (4) There is a fast (polylogarithmic time) algorithm which, given x, outputs $M(x)$. Note that, in particular, this property requires that $M(x)$

have size that is polylogarithmically bounded. We did not succeed in finding such a definition of good/bad.

Nevertheless, we were able to find a distributed filter by modifying the above approach. As in [6], we construct for each point x a set $Rep(x)$ of points that approximates $M(x)$ and can be found in polylogarithmic time. But our definition satisfies an additional property:

> (*)For every pair of points x, y with $x < y$, and for each $z \in Rep(x)$ there is a $z' \in Rep(y)$ with $z \leq z'$.

Having constructed such a set $Rep(x)$, we define $g(x)$ to be the maximum of $f(z)$ for $z \in Rep(x)$. It follows that g is monotone, agrees with f on all good points and can be computed in polylogarithmic time. Condition (*) provides a crucial difference between our definition and that in [6]. It means that the monotonicity of g is ensured without having to enforce it artificially by saving all of the past values. To enforce Condition (*), we are forced to redefine good/bad - in other words, the function g has to be different from f for more points. This redefinition has to be done very carefully to ensure that the error blowup is contained.

Finding a definition of good/bad and a suitable definition of $Rep(x)$ takes a significant amount of work. We start with a quickly testable notion of *accepted* and *rejected* points. The function f is monotone on all sound points, and the number of unsound points is small enough (in terms of ε_f). In [6], this notion of accepted/rejected serves as their definition of good/bad, but (for the reasons sketched above) it is not sufficient for our purposes.

The definition of *Rep* crucially uses a data structure of nested boxes (products of intervals). Ensuring property (*) is accomplished by as a careful and efficient "message passing" scheme which passes crucial information about the distribution of good points in a particular box to its sub-boxes.

2.2 A high level view of the filter

Our filter can be broken down into two procedures, *Sift* and *Build*. We specify below the main guarantees of these two procedures. We will fix a random seed ρ of $(d \log n)^{O(1)}$ length. These procedures are randomized but they will only use the random seed ρ. The guarantees of these procedures will hold with high probability. We show that given these procedures, we can easily construct a distributed filter satisfying the conditions of Theorem 2.1.1.

Given a function f on $[n]^d$, a subset $S \subseteq [n]^d$ is *f-admissible* if the restriction of f to S is monotone.

The first component of the filter, *Sift*, takes as input a point $x \in [n]^d$ and returns *accept* or *reject*. With high probability it satisfies the following properties:

S1 : The set of accepted points is f-admissible.

S2 : The set of rejected points has size at most $C_1(d)\varepsilon_f n^d$ where $C_1(d)$ is independent of n. (In our algorithm $C_1(d) = 2^{O(d^2)}$.)

S3 : *Sift* runs in time $(\log n)^{O(d)}$.

The second part of the filter, *Build*, takes as input a point $x \in [n]^d$ and (using *Sift*) returns a set $Rep(x)$ of *representative points* for x. This procedure satisfies:

B1 : Every point in $Rep(x)$ is less than or equal to x.

B2 : Every point in $Rep(x)$ is accepted by *Sift*.

B3 : For all x, y with $x \leq y$, for each $z \in Rep(x)$ there is a point $z' \in Rep(y)$ such that $z \leq z'$.

B4 : With high probability, the number of points x for which $x \notin Rep(x)$ is at most $C_2(d)$ times the number of points rejected by *Sift*, where $C_2(d)$ is independent of n. (In our algorithm, $C_2(d) = 2^{O(d^2)}$.)

B5 : *Build* runs in time $(\log n)^{O(d)} \times$ running time of *Sift*.

Our filter is then defined from *Build* as follows:

> Given x, $g(x) = \max\{f(z) : z \in Rep(x)\}$. If $Rep(x)$ is empty, then $g(x) = 0$.

Let us show that properties [S1]-[S3] and [B1]-[B5] ensure that the filter has the required properties of Theorem 2.1.1.

To see that g is monotone, let $x \leq y$. By the definition of g, $g(x) = f(z)$ for some $z \in Rep(x)$. By property [B3], there is a $z' \in Rep(y)$ such that $z \leq z'$. By [B2], both z and z' are accepted by *Sift* so by [S1], $f(z) \leq f(z')$. By the definition of $g(y)$ we have $g(y) \geq f(z')$ and by [S1], $f(z') \geq f(z)$, so $g(y) \geq f(z') \geq f(z) = g(x)$ as required.

To get an upper bound on the number of points for which $g(x) \neq f(x)$, note that $x \in Rep(x)$ implies $g(x) = f(x)$ and so by [B4], the number of points with $g(x) \neq f(x)$ is at most $C_2(d)$ times the number of points rejected by *Sift* which, by [S2] is at most $C_2(d)C_1(d)\varepsilon_f n^d$ as required.

Finally, the desired bound on the running time of the filter follows from [B5] and [S3].

2.3 The one-dimensional case

In this section, we describe our distributed monotonicity filter for the case $d = 1$.

2.3.1 The DAG $D(k)$ of intervals

For $x, y \in [n]$ we define the interval $[x, y]$ to be the set $\{z \in [n] : x \leq z \leq y\}$. If I is an interval we write $\min(I)$ for its smallest element and $\max(I)$ for its largest element; $\min(I)$ and $\max(I)$ are the *endpoints* of I and $I - \{\min(I), \max(I)\}$ is the

interior of I. If $\min(I) = 1$, we say I is *left extreme* and if $\max(I) = n$ we say I is *right extreme*.

Our filter makes use of a special set of intervals, which we now define. For integers $i \geq 1$ and $j \geq 0$, we define the interval

$$I_i^j = [j2^{i-1} + 1, (j+2)2^{i-1}].$$

The set $\{I_i^j : 1 \leq i \leq k, 0 \leq j \leq \frac{n}{2^{i-1}} - 2\}$ is denoted $\mathcal{I} = \mathcal{I}(k)$. The set $\mathcal{I}_i = \mathcal{I}_i(k)$, called the *ith level*, is the set of intervals $\{I_i^j \,|j \geq 0\}$. The ith level contains $\frac{n}{2^{i-1}} - 1$ intervals each of size 2^i.

An interval I_i^j is said to be *even* if j is even and *odd* if j is odd. Notice that the set of even intervals at level i comprise the natural partition of $[1, n]$ into $\frac{n}{2^i}$ intervals of size 2^i, while the odd intervals at level i partition the interval $[2^{i-1} + 1, n - 2^{i-1}]$ into $\frac{n}{2^i} - 1$ intervals of size 2^i.

We define a DAG $D = D(k)$ with vertex set $\mathcal{I}$, with an edge from interval I to interval J if $J \subseteq I$ and they belong to adjacent levels. The DAG $D(k)$ has k levels and $2^{k+1} - k - 2$ vertices ($2^{k+1-i} - 1$ at level i). The interval $I_n^0 = [1, n]$ is the unique interval of in-degree 0 and is called the *root* of D, and the intervals in level 1 (those having size 2) have out-degree 0 and are called *leaves*.

Figure 2.1 shows the the DAG $D(3)$ (with edges pointing downwards). Note that for $r < s$ the sub-DAG consisting of the first r levels of $D(s)$ is isomorphic to $D(r)$.

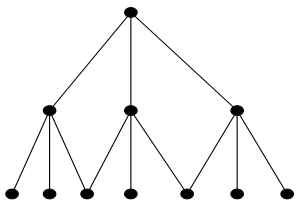

Figure 2.1: *The DAG* $D(3)$

Suppose I and J are intervals and there is an edge from I to J. We say that I is the *parent* of J and J is a *child* of I. We must have $|I| = 2|J|$. Furthermore, parent-child relationships fall into three categories:

- $\min(I) = \min(J)$. Here we say that J is the *left child* of I and I is the *right parent* of J.
- $\max(I) = \max(J)$. Here we say that J is the *right child* of I and I is the *left parent* of J.
- $\min(J) = \min(I) + |J|/2$ and $\max(J) = \max(I) - |J|/2$. Here we say that J is the *central child* of I and I is the *central parent* of J.

Every interval at level $i \geq 2$ has exactly 3 children, one left child, one central child and one right child.

Every non-root interval I has either one or two parents. If I is an odd interval, then it has one parent, and that parent is a central parent. If I is an even interval and neither left-extreme nor right-extreme then it has one left parent and one right parent. If I is left-extreme it has only a right parent, and if it is right-extreme it has only a left parent.

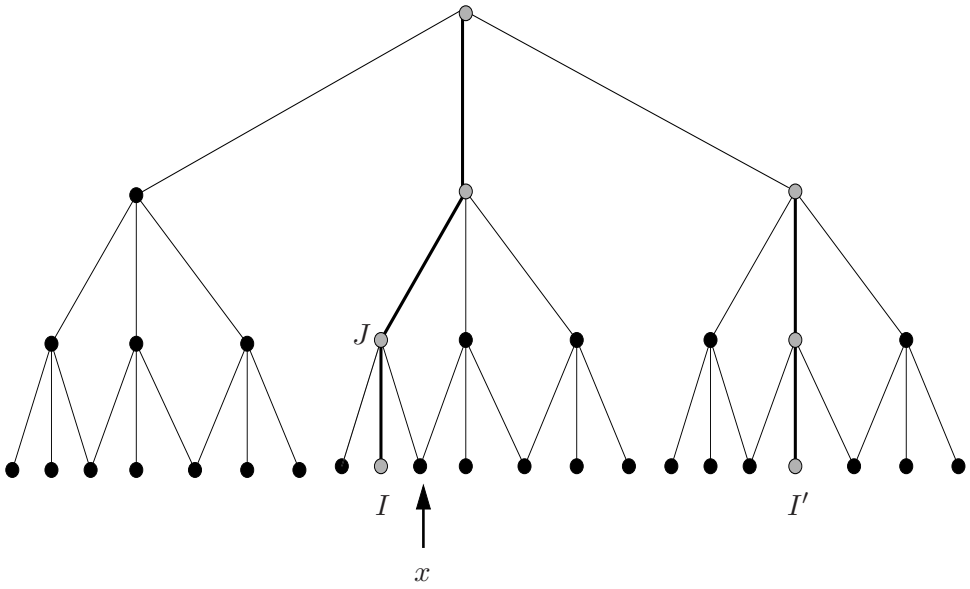

Figure 2.2: *Definitions*

For an interval I and point $x \in [n]$, we define :

- **Non-left parent of I:** Interval J is a *non-left parent* of I if it is either a right parent or a central parent of I. We observe that if I is right-extreme, then it has no non-left parent, but otherwise I has a *unique non-left parent*, which we denote by $nonleftpar(I)$. In Figure 2.2, interval J is the non-left parent of I.
- **Non-left path of I:** The *non-left path of* I, $path(I)$, is the unique sequence $I = I_1, I_2, \ldots, I_t$ where for $1 \leq j < t$, $I_{j+1} = nonleftpar(I_j)$ and I_t is right-extreme. In Figure 2.2, the non-left paths of I and I' are given by bold lines, and the intervals in them are shown by light colored circles.
- **Non-left ancestor of I:** Interval J is a *non-left ancestor* of I if $J \in path(I)$. Any light colored circle in the bold path leading from I to the root in Figure 2.2 is a non-left ancestor of I.
- **I is to the left of x:** I is *to the left* of point x if $y \leq x$ for all $y \in I$. In the figure, suppose x is in the interval shown by the arrow and not in I. Then I is to the left of x.

- I **is left-maximal for** x**:** I is *left-maximal* for x if I is left of x and x belongs to the interior of every non-left ancestor of I. For each x there are at most two intervals at each level that are left maximal for x. The intervals that are left-maximal for n are precisely those that are right-extreme. In the figure, the interval I is left-maximal for x.

- $upper(I)$: For I of size at least 4, we define $upper(I)$ to be the subinterval of I consisting of the greatest $|I|/4$ points of I. Note that if I is a non-left parent of an interval J, then every point in J is less than every point in $upper(I)$.

Some useful properties of $\mathcal{I}$ are stated below.

I1: Every interval in $\mathcal{I}_i$ has size 2^i.

I2: Every node of D has at most two ancestors at each level; in particular each point of $[n]$ belongs to at most two intervals at each level. It follows that every node has $O(\log n)$ ancestors.

I3: For $I_1, I_2 \in \mathcal{I}$, ff $I_1 \cap I_2 \neq \emptyset$ and $|I_1| < |I_2|$ then $I_1 \subset I_2$.

I4: For any two points $x < y$. there is an interval of $\mathcal{I}$ of size at most $4|[x, y]|$ containing both x and y.

I5: If $\mathcal{A}$ is a containment-free subset of $\mathcal{I}$ (i.e., no member of $\mathcal{A}$ contains another), then each point x belongs to at most 2 intervals of $\mathcal{A}$.

Properties [I1], [I2] and [I3] are obvious. For property [I4], let t be the integer such that $2^t \leq |[x, y]| < 2^{t+1}$, and let j be the largest integer such that $j2^{t+1} < x$. then $I^j_{t+2} = [j2^{t+1} + 1, (j + 2)2^{t+1}]$ contains x and y and has length $2^{t+2} \leq 4|[x, y]|$.

For property [I5], note that [I3] and $\mathcal{A}$ being containment-free, imply that all sets in $\mathcal{A}$ that contain x must have the same size; hence there are at most 2 of them.

We introduce another important definition.

- θ-dense : If S is a subset of $[n]$ and $\theta \in [0, 1]$, we say that S is θ-dense in I if $|S \cap I| \geq \theta|I|$. For $S \subseteq [n]$, define $\Lambda_\theta(S)$ to be the union of all $I \in \mathcal{I}$ such that S is θ-dense in I.

Lemma 2.3.1. *For any subset* S, $\Lambda_\theta(S) \leq \frac{2}{\theta}|S|$.

Proof. Let $\mathcal{J}$ be the subset of all $I \in \mathcal{I}$ for which S is θ-dense in I. Let $\mathcal{A}$ be the set of maximal members of $\mathcal{J}$. Then

$$|\bigcup_{I\in\mathcal{J}} I| = |\bigcup_{I\in\mathcal{A}} I| \leq \sum_{I\in\mathcal{A}} |I| \leq \frac{1}{\theta}\sum_{I\in\mathcal{A}} |S \cap I|.$$

Since $\mathcal{A}$ is a containment-free subcollection of $\mathcal{I}$, by property [I5], each element of S belongs to at most 2 members of $\mathcal{A}$. The final sum is at most $2|S|$, giving the upper bound of $\frac{2}{\theta}|S|$ on $|\bigcup_{I\in\mathcal{J}} I|$. □

2.3.2 The random seed

Our random seed will be interpreted as an n-ary string, i.e. a sequence selected independently from the set $[n]$. In the one-dimensional case, the seed is viewed as a pair of sequences $(s(1), \ldots, s(t))$ and $(r(1), \ldots, r(t))$ of length $t = c(\log n)^2$ elements of $[n]$, for some large enough constant c. Thus we use $2t \log n$ bits of randomness.

Given a number $s(i)$, we define $s(i)(\text{mod } I)$ as follows - suppose $I = [i, j]$. Then $s(i)(\text{mod } I) = i + s(i)(\text{mod } (j - i)) \in I$. Using these strings, we define for each interval I the sequences $s(\text{mod } I)$ and $r(\text{mod } I)$ in I^t, respectively to be $(s(1)\text{mod } I, \ldots, s(t)\text{mod } I)$ and $(r(1)\text{mod } I, \ldots, r(t)\text{mod } I)$.

2.3.3 The subroutine *Sift* in one dimension

The subroutine *Sift* takes as input a point x and outputs *accept* or *reject* in such a way that the set of accepted points is f-admissible. Anticipating what we need in the multidimensional case, we will define *Sift* in greater generality than we need for the one-dimensional case. We assume that the set of points is initially partitioned into sets *Eligible* and *Ineligible* points. This partitioning is provided by a subroutine which, on input x, tests whether $x \in Eligible$. We will show that *Sift* satisfies the following conditions:

S1′: Only eligible points are accepted and, with high probability, the set of accepted points is f-admissible.

S2′: The set of rejected points has size at most $C_1(\varepsilon_f n^d + |Ineligible|)$ for some constant C_1.

S3′: *Sift* runs in time $(\log n)^{O(1)}T$, where T is an upper bound on the time to test membership in *Eligible*.

When we use this subroutine for the one-dimensional filter we'll take $Eligible = [n]$ and $Ineligible = \emptyset$, but for the higher dimensional filter, we'll need the more general version.

Let us define a *violation* to be a pair (x, y) such that $x < y$ and $f(x) > f(y)$, or $x \in Ineligible$ or $y \in Ineligible$. We also say x *is a violation with* y (and vice versa).

Definition 2.3.2. *For a point x and $A \subseteq [n]$ -*

- *$violations(x, A)$ is the fraction of points in A that are in violation with x.*
- *For $\mu > 0$, x is* μ-sound *if $x \in Eligible$ and for all $I \in \mathcal{I}$ such that $x \in I$, $violations(x, I) < \mu$. Otherwise x is μ-unsound.*

The following lemma is similar to a result from [6].

Lemma 2.3.3. *1. Every μ-sound point belongs to Eligible.*

2. For $\mu \leq 1/8$, the set of μ-sound points is f-admissible.

3. *For $\mu \leq 1/2$, the number of μ-unsound points is at most $2\mu^{-1}(\varepsilon_f n + |Ineligible|)$.*

Proof. The first part is immediate from the definition of μ-sound.

For the second part, it suffices to show that for each violation (x, y) $(x < y)$ at least one of x and y is 1/8-unsound. This is immediate if either x or y belongs to $Ineligible$, so assume that $x, y \in Eligible$. Since $f(x) > f(y)$, for each $z \in [x, y]$ at least one of (x, z) and (y, z) is a violation. Thus $violations(x, [x, y]) + violiations(y, [x, y]) \geq 1$. By property [I4], there is an interval $I \in \mathcal{I}$ of size at most $4|[x, y]|$ that contains $[x, y]$, so $violations(x, I) + violations(y, I) \geq 1/4$, and so one of them is at least 1/8.

For the third part, by definition of ε_f, there is a subset S of size $\varepsilon_f n$ such that $[n] - S$ is f-admissible. By Lemma 2.3.1, $\Lambda_\mu(S \cup Ineligible)$ has size at most $2\mu^{-1}(\varepsilon_f n + |Ineligible|)$, so it suffices to show that every μ-unsound point x belongs $\Lambda_\mu(S \cup Ineligible)$. This is true for $x \in S \cup Ineligible$ since x belongs to an interval $I \in \mathcal{I}$ of size 2, and $S \cup Ineligible$ is μ-dense in I. If $x \notin S \cup Ineligible$, then the set of points violating x is a subset of $S \cup Ineligible$. Since x is μ-unsound there is an interval I containing x that contains at least $\mu|I|$ points violating x. Therefore $S \cup Ineligible$ is μ-dense in I and $x \in \Lambda_\mu(S \cup Ineligible)$. □

We define a subroutine **Violations** which takes as input a point x and interval $I \in \mathcal{I}$ and returns an estimate of $violations(x, I)$. This estimate is defined to be the fraction of points in the sample $s \mod I$ (as described in Section 2.3.2) that are violations with x. Clearly **Violations** runs in time $(\log n)^{O(1)}$. We say that **Violations** *fails* for x and I if $|\textbf{Violations}(x, I) - violations(x, I)| > 1/100$. We say that **Violations** *fails* if it fails for some x, I where $I \in \mathcal{I}$ and $x \in I$, and we say it succeeds otherwise.

$Sift(I)$: On input x, *Sift* rejects x if there exists some $I \in \mathcal{I}$ $(x \in I)$ such that **Violations**$(x, I) \geq 11/100$, and accepts x otherwise.

We say that *Sift* succeeds provided that:

- Every 1/10-sound point is accepted
- Every 1/8-unsound point is rejected.

Since there are $O(\log n)$ intervals of $\mathcal{I}$ containing x, and the sequence s of samples has length $t = (\log n)^{O(1)}$, we conclude that the running time of *Sift* is $(\log n)^{O(1)}$ (thus condition [S3] of Section 2.2 holds). From Lemma 2.3.3, we deduce that if *Sift* succeeds, then conditions [S1] and [S2] of Section 2.2 also hold.

Corollary 2.3.4. *If* Sift *succeeds then:*

1. *The set of points accepted by* Sift *is f-admissible.*
2. *The number of points rejected by* Sift *is at most $20(\varepsilon_f n + |Ineligible|)$.*

Finally, we observe that the probability that *Sift* fails is very small.

Lemma 2.3.5. *The probability that* Sift *fails is* $(n \log n)2^{-\Omega(t)}$*, where* t *is the length of the random sequence* s*.*

Note that for the given choice of t, this is $n^{-\omega(1)}$.

Proof. If *Sift* fails then there is a point x and $I \in \mathcal{I}$ containing x for which $|\textbf{Violations}(x, I) - \mathit{violations}(x, I)| \geq 1/100$. Fix x and I and let $p = \mathit{violations}(x, I)$. Then $\textbf{Violations}(x, I)$ is the average of t independent 0–1 random variables each with expectation p, and so by a standard Chernoff-type bound, $\Pr[|\textbf{Violations}(x, I) - \mathit{violations}(x, I)| \geq 1/100] \leq 2^{-\Omega(t)}$. A union bound over the $O(n \log n)$ pairs (x, I) gives the lemma. □

2.3.4 The procedure *Build*: 1-dimensional case

We now turn to the description of the procedure *Build* (for the 1-dimensional case). We will use the procedure *Sift* with the trivial subroutine *Eligible* that accepts every point.

Recall that *Build* is supposed to take as input a point x and return a set $\mathit{Rep}(x)$ consisting of (some) eligible points less than or equal to x. The main ingredients are a pair of procedures Sample and Refine. Each takes as input an interval I and returns a small subset of points of I. For a set S, let $\mathit{Sift}(S)$ denote the set of points of S accepted by *Sift*. We now describe the procedures Sample and Refine.

$\mathit{Sample}(I)$**:** Consider t, as defined in Section 2.3.2. If $|I| \leq t$, $\mathit{Sample}(I) = I$. If $|I| > t$, $\mathit{Sample}(I)$ is the set of points in the sequence $r \mod I = (r(1) \mod I, \ldots, r(t) \mod I)$, as described in Section 2.3.2.

$\mathit{Refine}(I)$**:** On input I, determine $\mathit{Sample}(J)$ for each $J \in \mathit{path}(I)$. If $\mathit{Sift}(\mathit{Sample}(J)) \cap \mathit{upper}(J) \neq \emptyset$ for all $J \in \mathit{path}(I)$ then $\mathit{Refine}(I) = \mathit{Sift}(\mathit{Sample}(I))$. Otherwise, $\mathit{Refine}(I) = \emptyset$.

Claim 2.3.6. *If* $\mathit{Refine}(I)$ *is nonempty then* $\mathit{Refine}(J) \cap \mathit{upper}(J) \neq \emptyset$ *for all non-left ancestors* J *of* I*.*

Proof. We first show that if $\mathit{Refine}(I)$ is nonempty then $\mathit{Refine}(J) = \mathit{Sift}(\mathit{Sample}(J))$ for all non-left ancestors J of I. Consider some interval I at depth d such that $\mathit{Refine}(I)$ is nonempty. By definition, $\mathit{Sift}(\mathit{Sample}(I')) \cap \mathit{upper}(I') \neq \emptyset$ for all $I' \in \mathit{path}(I)$ and $\mathit{Refine}(I) = \mathit{Sift}(\mathit{Sample}(I))$. Note that for non-left ancestor J, $J \in \mathit{path}(I)$. Obviously, $\mathit{Sift}(\mathit{Sample}(J')) \cap \mathit{upper}(J') \neq \emptyset$ for all $J' \in \mathit{path}(J)$ and $\mathit{Refine}(J) = \mathit{Sift}(\mathit{Sample}(J))$.

For all non-left ancestors J of I, $\mathit{Sift}(\mathit{Sample}(J)) \cap \mathit{upper}(J) \neq \emptyset$. Therefore, $\mathit{Refine}(J) \cap \mathit{upper}(J) \neq \emptyset$. □

Finally we define the procedure *Build*.

$\mathit{Build}(x)$**:** On input x, the output $\mathit{Rep}(x)$ is defined to be the union of $\mathit{Refine}(I)$ over all intervals I that are left-maximal with respect to x.

For the sake of definition and implementation, we can redefine $Refine(I)$ to simply be the largest point in the present $Refine(I)$. We describe $Refine(I)$ as a set for two reasons - (1) to make the proof of [B3] more convenient, (2) to prepare for the extension to the multi-dimensional case where $Refine(I)$ can not be taken to be a single point. We now verify properties [B1]-[B5] of Section 2.2.

[B1] and [B2]: By definition of $Rep(x)$, all points in $Rep(x)$ are less than or equal to x, and each point in $Rep(x)$ is accepted by *Sift*.

[B3]: Let x, y be arbitrary points in $[n]$ with $x \leq y$ and let $z \in Rep(x)$. We must show that there is a $z' \in Rep(y)$ with $z \leq z'$. Since $z \in Rep(x)$, there is an interval I that is left-maximal for x such that $z \in Refine(I)$. If I is left-maximal for y then we take $z' = z$. Otherwise, let J be the largest interval in $path(I)$ that is to the left of y ; by definition J is left maximal for y. Since $Refine(I) \neq \emptyset$, $Refine(J) \cap upper(J) \neq \emptyset$ (by Claim 2.3.6) so we can select $z' \in Refine(J) \cap upper(J)$. Since J is a non-left ancestor of I, every point in I is less than every point in $upper(J)$ (this follows by an easy induction on $|J|$), so $z \leq z'$.

[B4]: We bound the number of points for which $x \notin Rep(x)$. We say that *Sample* fails for interval I if at least half the points of $upper(I)$ are accepted by *Sift*, but no points in $Sample(I) \cap upper(I)$ are accepted by *Sift*. We say that *Sample* fails if *Sample* fails for some interval I of size at least 4, and *Sample* succeeds if it succeeds for all intervals I of size at least 4, that is, if for each such interval I, if at least $|I|/8$ points of $upper(I)$ are accepted by *Sift*, then $Sift(Sample(I) \cap upper(I)) \neq \emptyset$.

We will prove shortly that *Sample* succeeds with high probability. Assuming that *Sample* succeeds, we prove an upper bound on the number of points x for which $x \notin Rep(x)$. For each $x \in [2, n]$, the interval $[x-1, x]$ is left-maximal for x. If $x \notin Rep(x)$ then $x \notin Refine([x-1, x])$. Therefore either *Sift* rejects x or there is a non-left ancestor J of $[x-1, x]$ such that $Sift(Sample(J)) \cap upper(J) = \emptyset$. Since *Sample* succeeds, fewer than half the points in $upper(J)$ are accepted by *Sift*, which means at least $1/8$ of the points of J are rejected by *Sift*. Letting *Reject* be the set of points rejected by *Sift*, the set of points for which $x \notin Rep(x)$ is a subset of $\Lambda_{1/8}(Reject)$. By Lemma 2.3.1, the number of such points is at most $16|Reject|$.

Proposition 2.3.7. *Sample succeeds with probability* $> 1 - n^{O(1)}2^{-\Omega(t)} = 1 - n^{-\omega(1)}$.

Proof. We first give an upper bound on the probability that *Sample* fails for a fixed interval I. If fewer than half the points of $upper(I)$ are accepted by *Sift*, then *Sample* succeeds on I. So assume that at least half the points of $upper(I)$ are accepted by *Sift*. Then for each point in $Sample(I)$, the probability that it is one of the points in $upper(I)$ accepted by *Sift* is at least $1/8$. By the independence of the samples in $Sample(I)$, we can apply a simple Chernoff bound argument to show that $\Pr[Sift(Sample(I) \cap upper(I)) = \phi] = 2^{-\Omega(t)}$. A union bound over all intervals and points completes the proof. □

[B5]: The running time of *Build* is bounded as follows. For any x, there are at

most $2 \log n$ intervals (at most 2 on each level) that are left-maximal with respect to x. For each such I, we compute $Refine(I)$ which involves computing $Sample(J)$ for each of the at most $\log n$ non-left ancestors J of I and calling *Sift* for each of the points in $Sample(J)$. Since the size of $Sample(J)$ is $t = O((\log n)^2)$ the running time of *Build* is at most the cost of $O((\log n)^4)$ calls to *Sift*. (No attempt has been made to optimize the exponent of $\log n$ in this description.)

This completes the description and proof of correctness for the filter in the 1-dimensional case.

2.4 A filter for multidimensional data

2.4.1 Boxes and lines

For $x, y \in [n]^d$, $[x, y]$ denotes the set $\{z : \ x \leq z \leq y\}$. This set is a product of (1-dimensional) intervals $[x_1, y_1] \times \cdots \times [x_d, y_d]$ and is called a *box*. For a box $B = [x, y]$, we write $B = B_1 \cdots \times \cdots B_d$ where $B_r = [x_r, y_r]$ is the interval obtained by projecting B onto the rth coordinate axis.

A box B is *degenerate* in direction r if $|B_r| = 1$, *non-degenerate* in direction r if $|B_r| > 1$, and *spanning* in direction r if $B_r = [1, n]$.

An *r-line* is a box that is spanning in dimension r and degenerate in every other dimension. The r-lines partition $[n]^d$ into n^{d-1} sets, each of size n. We say that $x \leq_r y$ if $x \leq y$ and x, y lie in the same r-line. The r-line passing through x is denoted by $x^{(r)}$.

There is a natural bijection between an r-line L and the set $[n]$ given by $x \in L \leftrightarrow x_r$. For $j \in [n]$ we write j_L for the corresponding point on L, and for $S \subseteq [n]$ we write S_L for the corresponding subset of L. Define $\mathcal{I}_L$ to be the set of I_L for $I \in \mathcal{I}$, where $\mathcal{I} = \mathcal{I}(k)$ as defined in Subsection 2.3.1.

2.4.2 The random seed

The random seed (which consists of independent uniformly random elements selected from $[n]$) is divided into $2d$ sequences $s^1, \ldots, s^d, r^1, \ldots, r^d$, each of length $t(d) = cd(\log n)^2$ (for some constant c).

2.4.3 The function *Sift*, multi-dimensional case

We define some auxiliary procedures based on which *Sift* will be constructed.

Linesift$_j(x)$: For $j \in [d]$ - On input x, *Linesift*$_j$ runs the one-dimensional *Sift* on x with respect to the j-line $x^{(j)}$, using the random sample s^j. As with *Sift*, *Linesift*$_j$ requires an auxiliary procedure $Eligible_j$, which we assume does not use the random sample s^j.

For each j-line L, the analysis of the one-dimensional *Sift* applies to *Linesift*$_j$. For each line L, one defines the notion of μ-soundness of a point $x \in L$ with respect to the line L in the obvious way. We say that *Linesift*$_j$ succeeds for a j-line L if -

- Every point that is 1/10-sound point with respect to L is accepted
- Every point that is 1/8-unsound with respect to L is rejected.

We say that $Linesift_j$ succeeds if it succeeds on every j-line. By Lemma 2.3.5, the probability that $Linesift_j$ fails for a particular j-line is bounded above by $n \log n 2^{-\Omega(t)}$. Therefore the probability that $Linesift_j$ fails on some j-line is bounded above by $n^d \log n 2^{-\Omega(t)}$. For the selected $t = cd(\log n)^2$ this is $n^{-\omega(d)}$.

This holds for any choice of the auxiliary procedure $Eligible_j$, provided that $Eligible_j$ does not depend on the random string s^j.

$Sift_j(x)$: For $0 \leq j \leq d$ - $Sift_0$ accepts every input point x. For $0 \leq j \leq d$, $Sift_j$ is obtained by running $Linesift_j$, taking $Eligible_j$ to be $Sift_{j-1}$.

$Sift(x)$: This is defined to be $Sift_d$.

Let $Accepted_j$ (resp., $Rejected_j$) be the set of points accepted (resp., rejected) by $Sift_j$. Let $\mathcal{C}_j$ be the partition of $[n]^d$ into n^{d-j} classes, where points are assigned to classes according to their last $d-j$ coordinates. Two points x, y in the same j-line with $x <_j y$ form a j-violation if $f(x) > f(y)$ or one of them is in $Ineligible_j = Rejected_{j-1}$.

Lemma 2.4.1. *Assume that* Linesift_1, Linesift_2, ..., Linesift_d *all succeed.*
For each $j \in \{0, \ldots, d\}$,

1. *For each* $C \in \mathcal{C}_j$, *the set* $C \cap Accepted_j$ *is* f*-admissible.*
2. $|Rejected_j| \leq (20^{j+1} - 20)/(19)\varepsilon_f n^d$,
3. *Let* T_j *be the running time of* Sift_j. *Then* $T_j \leq (\log n)^{O(j)}$.

Proof. We proceed by induction on j; the case $j = 0$ is trivial.

Part 1 : Assume $j \geq 1$ and that the lemma is true for $j-1$. Let $C \in \mathcal{C}_j$; we want to show that $C \cap Accepted_j$ is f-admissible. Consider $y, z' \in C$ with $y < z'$ and $f(y) > f(z')$; we want to show that at least one of them is not in $Accepted_j$. This is clear if either one is in $Rejected_{j-1}$, so we may assume that y, z' are both in $Accepted_{j-1}$.

Let L be the j-line through y and L' be the j-line through z' (possibly $L = L'$.) For $x \in L$, let $x' \in L'$ be the point such that $x_j = x'_j$. We denote by z the point on L such that $z_j = z'_j$. Let $S \subseteq L$ be the interval of points $[y, z]$ and S' be the corresponding interval $[y', z']$. By property [I4], there is an interval $I \subseteq L$ that corresponds to an interval of $\mathcal{I}$, contains S and has size at most $4|S|$. Let I' be the corresponding interval in L'.

We claim that for each $w \in S$, w is a violation of y or w' is a violation with z' (see Figure 2.3). If $f(y) > f(w)$ or $f(w') > f(z')$ again we are done, so assume $f(y) \leq f(w)$ and $f(w') \leq f(z')$, then we have $f(w') < f(w)$. The points w and w' belong

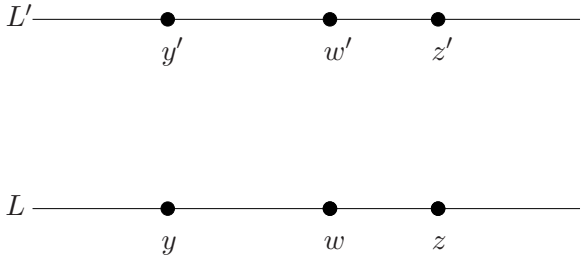

Figure 2.3: The points y, z, w, y', z', w' in proof of Lemma 2.4.1, Part 1

to the same class $B \in \mathcal{C}_{j-1}$, and by the induction hypothesis $Accepted_{j-1} \cap B$ is f-admissible which means that at least one of w and w' is in $Rejected_{j-1} \subseteq Rejected_j$, which completes the proof of the claim.

It follows that $violations(y, S) + violations(z', S') \geq 1$ and so $violations(y, I) + violations(z', I') \geq 1/4$ and so one of these is at least $1/8$. This implies that one of y or z' is $1/8$-unsound with respect to L, and so under the assumption that $Linesift_j$ succeeds, one of them is rejected. This completes the proof of the first part.

Part 2 : There is a subset S of $[n]^d$ of size $\varepsilon_f n^d$ such that $[n]^d - S$ is f-admissible. Let point $x \in Rejected_j - (S \cup Rejected_{j-1})$. The point x must be $1/10$-unsound with respect to the j-line $L = x^{(j)}$. Therefore, there is some interval $I \subseteq L$ corresponding to an interval in $\mathcal{I}$ containing x such that $S \cup Rejected_{j-1}$ is $1/10$-dense in I. For each j-line ℓ, let $\varepsilon(\ell) = |S \cap \ell|/n$ and let $R_{j-1}(\ell) = Rejected_{j-1} \cap \ell$ and $R_j(\ell) = Rejected_j \cap \ell$. By Lemma 2.3.1, $|R_j(\ell)| \leq 20(\varepsilon(\ell)n + |R_{j-1}(\ell)|)$. Summing over all j-lines, we get $|Rejected_j| \leq 20(\varepsilon_f n^d + |Rejected_{j-1}|)$. Using the induction hypothesis, we complete the proof of the second part.

Part 3 : The procedure $Sift_j$ makes $(\log n)^{O(1)}$ calls to $Sift_{j-1}$ (refer to Section 2.3.3). A simple induction completes the proof. □

Corollary 2.4.2. *With high probability :*

1. *The function f restricted to the set of points accepted by* Sift *is monotone.*
2. *The number of points rejected by* Sift *is at most $20^{d+1}\varepsilon_f n^d$.*

2.4.4 The DAG $\Delta^d(k)$

Before describing the function $Build$, we need some additional definitions.

We consider the set $\mathcal{B} = \mathcal{B}(k)_d$ to be the set of of all boxes of the form $B = B_1 \times \cdots B_d$ where each $B_j \in \mathcal{I}(k)$ (the set of intervals defined for the one-dimensional case). For each $r \in [d]$, we define an equivalence relation on $\mathcal{B}$: for $B, C \in \mathcal{B}$, $B \sim_r C$ if $B_j = C_j$ for all $j \neq r$. For each r-equivalence class $\mathcal{C}$, the mapping taking $B \in \mathcal{C}$ to B_r is a bijection between $\mathcal{C}$ and $\mathcal{I}(k)$.

We define a DAG $\Delta = \Delta^d(k)$ on vertex set $\mathcal{B}$ as follows: (B, C) is in Δ if and only if for some $r \in [d]$ $B \sim_r C$ and $(B_r, C_r) \in D$, where D is the DAG defined

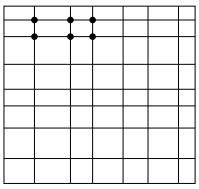

Figure 2.4: $Sample(B)$ and $Lines(B)$

for the one-dimensional case. In this case we say that B is an r-parent of C. We adapt the terminology from the one-dimensional case, If B is an r-parent of C we say that C is the (left,right,central) r-child of B if C_r is the (left,right,central) child of B_r, and we say that B is the (left,right,central,non-left) r-parent of C if B_r is the (left,right,central,non-left) parent of C_r.

For a point x and box B we say that B is to the left of x if for each $j \in [d]$, B_j is to the left of x_j. We say that B is left-maximal for x if for each $j \in [d]$, B_j is left-maximal for x_j.

We say that a box C is a non-left ancestor of B if for all $j \in [d]$, C_j is a non-left ancestor of B_j with respect to the one-dimensional dag D. Intuitively, a non-left ancestor of B is obtained by repeatedly taking a non-left parent along some direction. The number of non-left ancestors of B is $(\log n)^{O(d)}$, since any interval in $\mathcal{I}$ has at most $O(\log n)$ non-left ancestors.

2.4.5 The function *Build* : multi-dimensional case

We now turn to the description of the procedure *Build*. We will make use of the multi-dimensional version of *Sift*.

Recall that *Build* is supposed to take as input a point x and returns a set $Rep(x)$ of points. Similar to the one-dimensional case, *Build* will use a pair of procedures $Sample$ and $Refine$, each of which take as input a box $B \in \mathcal{B}$ and returns a small subset of points of B. The procedure $Refine$ takes as input a box B and only returns points that are accepted by *Sift*.

$upper(S)$: This is the set consisting of the largest $|S|/4$ points in a segment S of size at least 4.

$Sample(B)$: Remember that $t = cd(\log n)^2$, for sufficiently large constant c. If $|B| \leq t$, $Sample(B) = B$. If $|B| > t$, $Sample(B)$ is defined to be the product set $Sample(B) = \prod_{i=1}^{d} Sample_i(B)$ where $Sample_i(B)$ is the set of points of the form $r^i(j) \mod B_i$, where $1 \leq j \leq t$. Refer to Figure 2.4. The points in $Sample(B)$ are the points of some kind of a random grid.

$Lines(B)$: This is the set of all lines L (in one of the d coordinate directions) such that $|L \cap B| \geq 4$ and $L \cap Sample(B) \neq \emptyset$. In Figure 2.4, each of the lines shown are in $Lines(B)$.

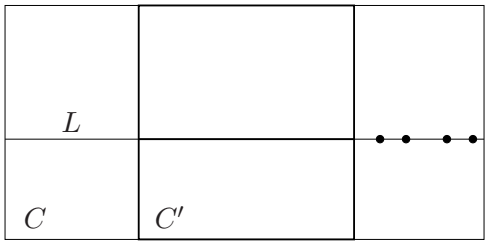

Figure 2.5: $Refine$ and $BadBoxes$

We will define two procedures which take as input a box B, $Refine(B)$ and $BadLines(B)$ (abusing notation, these will also denote the outputs). The outputs will be of the form $Refine(B) \subseteq Sample(B)$, and $BadLines(B) \subseteq Lines(B)$.

$Refine(B)$: This is the set of $x \in Sample(B)$ that are (i) accepted by *Sift*, and (ii) not contained in any line L that is in $BadLines(P)$ for some non-left parent P of B.

$BadLines(B)$: This is the set of those lines $L \in Lines(B)$ of size at least 4 such that $upper(L \cap C) \cap Refine(B)$ is empty. Note that $BadLines(B)$ contains $BadLines(P)$, for any non-left parent of B.

Given B, if $BadLines(P)$ is computed for all non-left parents P of B, then $Refine(B)$ can be computed. If $Refine(B)$ has been computed, then $BadLines(B)$ can be computed. Both these procedures can be computed by consider all non-left ancestors C of B in non-increasing order of size. Refer to Figure 2.5. The bigger box is C and it has a child C'. The line L is in $Lines(C)$. If none of the sample points in $upper(L \cap C)$ (shown by dark circles) are in $Refine(C)$, then L is in $BadLines(C)$. As a result, no point in $L \cap C'$ is in $Refine(C')$.

Build(x) : On input x, the output $Rep(x)$ is defined to be the union of $Refine(B)$ over all boxes that are left-maximal for x.

We now verify properties [B1]-[B5] of Section 2.2.

[B1] and [B2]: Properties [B1] and [B2], that all points in $Rep(x)$ are less than or equal to x, and that each point in $Rep(x)$ is accepted by *Sift*, are immediate from the definition of $Rep(x)$.

[B5]: We bound the running time of *Build*. There are at most $(\log n)^{O(d)}$ boxes that are left-maximal for x. For each such box B we need to compute $Refine(B)$. For this we must compute $Refine(C)$ and $BadLines(C)$ for each of the at most $(\log n)^{O(d)}$ non-left ancestors C of B. For this, we need to look at each of the at most $(\log n)^{O(d)}$ points $y \in Sample(C)$, and apply *Sift* (which takes time $(\log n)^{O(d)}$),

and test whether y belongs to one of the at most $(\log n)^{O(d)}$ lines in $BadLines(C')$ for some parent C' of C. The overall running time is therefore $(\log n)^{O(d)}$.

[B3] : Let x, y be arbitrary points in $[n]^d$ with $x \leq y$ and let $z \in Rep(x)$. We must show that there is a $z' \in Rep(y)$ with $z \leq z'$. It suffices to consider the case that x and y differ only in one coordinate, say coordinate j, since the general case will then follow by an easy induction on the number of coordinates in which x and y differ.

Since $z \in Rep(x)$, there is a box B with $z \in Refine(B)$, such that B is a left-maximal for x. Define the box C such that $C_i = B_i$ for $i \neq j$ and C_j is equal to the largest non-left ancestor of B_j that is left of y_j. It follows that C is left-maximal for y.

Let L be the j-line through z. We claim that $upper(L \cap C) \cap Refine(C) \neq \emptyset$; if so we can select z' to be any element of $upper(L \cap C) \cap Refined(C)$ and then $z \leq z'$ and $z' \in Rep(y)$. Suppose for contradiction, $upper(L \cap C) \cap Refine(C) = \emptyset$, then by definition L belongs to $BadLines(C)$. A simple induction shows that for every j-descendant C' of C, $L \cap Refine(C') = \emptyset$, but this contradicts that $z \in B$ (since B is a j-descendant of C).

[B4]: This is the hardest part and we break it into a separate subsection.

2.4.6 Proof that *Build* satisfies [B4]

We want to get an upper bound on the number of points x for which $x \notin Rep(x)$ that holds with high probability. We assume throughout this analysis that the random strings $s^1, \ldots, s^d$ used for *Sift* are fixed in such a way that *Sift* succeeds (in the sense defined earlier.)

Let $A(x)$ be the box $[x_1 - 1, x_1] \times [x_2 - 1, x_2] \times \cdots \times [x_d - 1, x_d]$. $A(x)$ is left-maximal for x and so $Refine(A(x)) \subseteq Rep(x)$. So it suffices to prove an upper bound on the number of x for which $x \notin Refine(A(x))$.

For each box C, we define $Lines'(C)$, $Refine'(C)$ and $BadLines'(C)$ in a way that parallels $Lines(C)$, $Refine(C)$ and $BadLines(C)$, the key difference being that we look over all points in C not just those in $Sample(C)$. $Lines'(C)$ is the set of all lines L such that $|L \cap C| \geq 4$. $Refine'(C)$ is defined to be the set of $x \in C$ that are (i) accepted by *Sift*, and (ii) not contained in any line L that is in $BadLines'(P)$ for some non-left parent P of C. $BadLines'(C)$ is the set of those lines $L \in Lines'(C)$ such that $|L \cap C| \geq 4$ and $|L \cap (C - Refine'(C))| \geq |L \cap C|/8$.

Observe that, with the random strings $s^1, \ldots, s^d$ being fixed, the functions $Refine'$, $Lines'$ and $BadLines'(P)$ are deterministic.

We say that *Build* succeeds if for all boxes B, $BadLines(B) \subseteq BadLines'(B)$.

Lemma 2.4.3. *The probability that* Build *fails is at most* $n^{O(d)}2^{-\Omega(t)}$.

Proof. Let us say that a box-line pair (B, L) is *relevant* if $|L \cap B| \geq 4$ and $|upper(L \cap B) \cap Refine'(B)| > |L \cap B|/8$. For each relevant box-line pair (B, L), define the

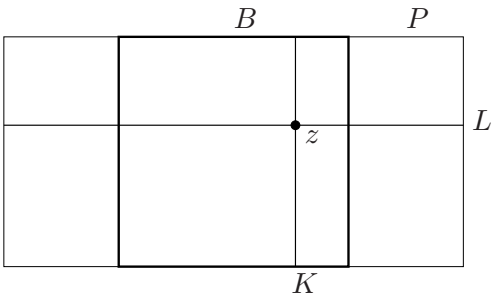

Figure 2.6: Why $F(B, L)$ occurs

event $F(B, L)$ to be the event that $Sample(B)$ contains no points of $upper(L \cap B) \cap Refine'(B)$. We prove the following claim.

Claim 2.4.4. *If* Build *fails then one of the events $F(B, L)$ happens.*

Proof. Suppose that *Build* fails. Let B be a box of maximum size for which $BadLines(B) \not\subseteq BadLines'(B)$. Let $L \in BadLines(B) - BadLines'(B)$. We show that (B, L) is a relevant pair and $F(B, L)$ occurred.

Since $L \notin BadLines'(B)$, $|L \cap (B - Refine'(B))| < |L \cap B|/8$. Therefore, $|L \cap B \cap Refine'(B)| \geq 7|L \cap B|/8$ and $|upper(L \cap B) \cap Refine'(B)| \geq |L \cap B|/8$, implying that (B, L) is relevant.

Suppose $F(B, L)$ does not occur. Then $Sample(B) \cap upper(L \cap B) \cap Refine'(B) \neq \emptyset$. Let $z \in Sample(B) \cap upper(L \cap B) \cap Refine'(B)$. Refer to Figure 2.6. If $z \in Refine(B)$, then $z \in Refine(B) \cap upper(L \cap B)$ contradicting that $L \in BadLines(B)$. So $z \notin Refine(B)$. Since $z \in Refine'(B)$ it must be accepted by *Sift*. Since z is accepted and also belongs to $Sample(B) - Refine(B)$ it must be that z belongs to a line K that is in $BadLines(P)$ for some parent P of B. Again, refer to Figure 2.6. Note that K does not have to be aligned with L. But by the maximality of $|B|$, $K \in BadLines'(P)$ which contradicts $z \in Refine'(B)$. Therefore if none of the events $F(B, L)$ happen then *Build* succeeds. □

So now we prove an upper bound on the probability of $\bigcup F(B, L)$ over all relevant pairs (B, L). We will use a union bound; the number of relevant pairs is bounded above by $n^{O(d)}$, so we need to prove a $2^{-\Omega(t)}$ bound on the probability of $F(B, L)$ where (B, L) is a relevant pair. It is important to observe that for each such pair the set $upper(L \cap B) \cap Refine'(B)$ and the event $F(B, L)$ does not depend on the random seed r (though they do depend on s), and therefore $Sample(B)$ is independent of the set $upper(L \cap B) \cap Refine'(B)$. For relevant (B, L) the probability that $Sample(B)$ is disjoint from $upper(L \cap B) \cap Refine'(B)$ is at most $(7/8)^t$ and thus the probability that *Build* fails is $n^{O(d)}(7/8)^t$. □

Proposition 2.4.5. *If* Build *succeeds then for each point x, $Refine'(A(x)) \subseteq Refine(A(x))$.*

Proof. $Sample(A(x)) = A(x)$, and therefore $y \notin Refine(A(x))$ means that y is rejected by *Sift* or y belongs to $BadLines(P)$ for some non-left parent of $A(x)$. Since *Build* succeeds, $BadLines(P) \subseteq BadLines'(P)$ and so $y \notin Refine'(A(x))$. □

So now it remains to find an upper bound on the size of $\cup_x (A(x) - Refine'(A(x))$.

In Section 2.3.1, we defined $\Lambda_\theta(S)$ to be the union of all intervals $I \in \mathcal{I}$ for which $|S \cap I|/|I| \geq \theta$. We adapt this definition to the multidimensional setting.

$\Lambda^j_\theta(S)$: For $j \in [d]$, let $\Lambda^j_\theta(S)$ be the union over all j-lines L, of the union of all $I \in \mathcal{I}(L)$, for which $|S \cap I|/|I| \geq \theta$.

$\Upsilon^k_\theta(S)$: Let $\Upsilon_\theta(S) = \bigcup_{j \in [d]} \Lambda^j_\theta(S)$, and $\Upsilon^k_\theta(S)$ is defined by $\Upsilon^1_\theta(S) = \Upsilon_\theta(S)$ and $\Upsilon^k_\theta(S) = \Upsilon_\theta(\Upsilon^{k-1}_\theta(S))$.

We will show:

Lemma 2.4.6. *$\bigcup_x A(x) - Refine'(x) \subseteq \Upsilon^d_\theta(S)$ where S is the set of points rejected by* Sift *and $\theta = 2^{-(d+2)}$.*

Assuming the lemma, we finish the proof of property [B4]. By Lemma 2.3.1, $\Lambda^j_\theta(S) \leq (2/\theta)|S|$, and therefore $\Upsilon_\theta(S) \leq (2d/\theta)|S|$. Thus $\Upsilon^d_\theta(S) \leq (2d/\theta)^d|S|$ which for the given value of θ is $2^{O(d^2)}$ times the number of points rejected by *Sift*.

So it remains to prove Lemma 2.4.6. We first need some additional definitions.

Cylinder: A cylinder is a box that for each direction j, is either degenerate or full. We say that C is a J-cylinder if J is the set of full directions.

Hyperplane: For $j \in [d]$, an $([n] - \{j\})$-cylinder is also called a *j-hyperplane.* For $r \in [n]$ we write $H_j(r)$ for the j-hyperplane consisting of all points x with $x_j = r$.

Cylinder type: Note that there is a natural bijection between a J-cylinder C and the set $[n]^{|J|}$. Using this bijection, we can define a family of boxes $\mathcal{B}(C)$ contained in C and a digraph $\Delta(C)$. Thus each box $B \in \mathcal{B}(C)$ has the form $B_1 \times \cdots \times B_d$ where $B_j = C_j$ is a singleton for $j \notin J$ and $B_j \in \mathcal{I}$ for $j \in J$. Let C be the unique cylinder such that $B \in \mathcal{B}(C)$, which is also the unique smallest cylinder containing B. We call C the *cylinder-type* of B.

Dimensionality: The *dimensionality* of a box B, $dim(B)$ is the number of non-degenerate directions.

***BadBoxes*:** This is a set of boxes. Consider boxes in increasing order of dimensionality. For boxes of the same dimensionality, we consider them in decreasing order of size. Initially we consider 0-dimensional boxes (points) and declare such a box to be bad if and only if the point is rejected by *Sift*. For a box B of dimension at least 1, we put $B \in BadBoxes$ if either of the following hold -

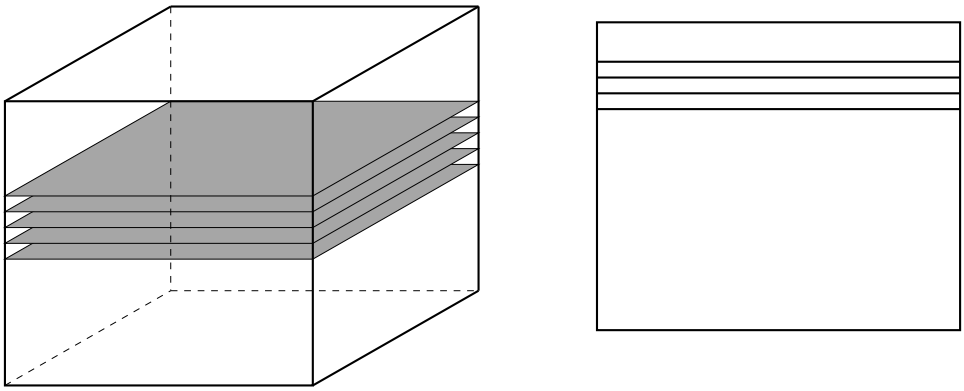

Figure 2.7: $BadBoxes$

1. There is a box B' of the same cylinder type as B that contains B and is in $BadBoxes$. (Note that B' is considered before B.)
2. There is a non-degenerate direction j of B such that for at least $1/2^{d+2}$ of the values $r \in B_j$, $H_j(r) \cap B \in BadBoxes$. (Note that each box $H_j(r) \cap B$ has lower dimension than B and is considered before B).

Refer to Figure 2.7. For the two-dimensional box on the right, the boxes $H_j(r) \cap B$ could look like the lines. If a large fraction of the lines are in $BadBoxes$, then B is in $BadBoxes$. Suppose B is a three-dimensional box, as shown in the figure. Then if many of the gray planes shown are in $BadBoxes$, then B is in $BadBoxes$.

Proof of Lemma 2.4.6. Observing that $A(x)$ is a full-dimensional box, the lemma will follow from:

P1 For any full-dimensional box $B \in \mathcal{B}$, if $x \in B - Refine'(B)$ there is a cylinder C containing x such that that $B \cap C \in BadBoxes$.

P2 Every $B \in BadBoxes$ is contained in $\Upsilon^d_\theta(S)$

Proof of [P1] : We prove [P1] by reverse induction on $|B|$. If $B = [n]^d$, then $x \in B - Refine'(B)$ iff x is rejected by *Sift*. We take C to be the singleton cylinder $\{x\}$ and $\{x\} = B \cap C$ belongs to $BadBoxes$.

Suppose $|B| < n^d$. If x is rejected by *Sift*, we take $C = \{x\}$. Otherwise, since $x \in B - Refine'(B)$, there is a parent P of B and line $L \in BadLines'(P)$ with $x \in P \cap L$. Let j be the direction of L. Refer to Figure 2.8.

Claim 2.4.7. *There is a cylinder C containing L such that $C \cap P \in BadBoxes$.*

If this is true, then $C \cap B$ has the same cylinder type as $C \cap P$. Therefore $C \cap B$ is in $BadBoxes$, and C certainly contains x, completing the proof of [P1].

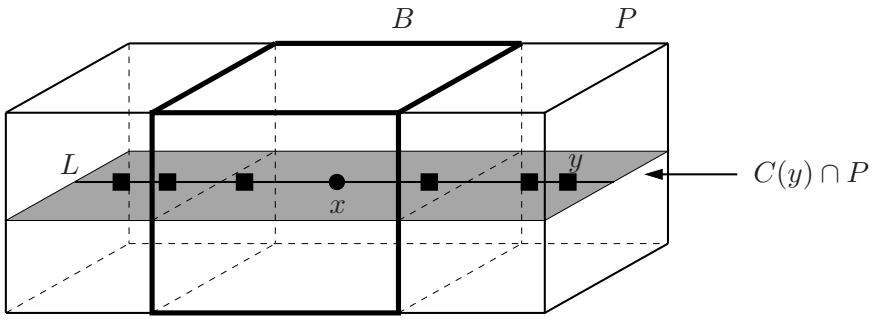

Figure 2.8: Case 1 of Proof of [P1]

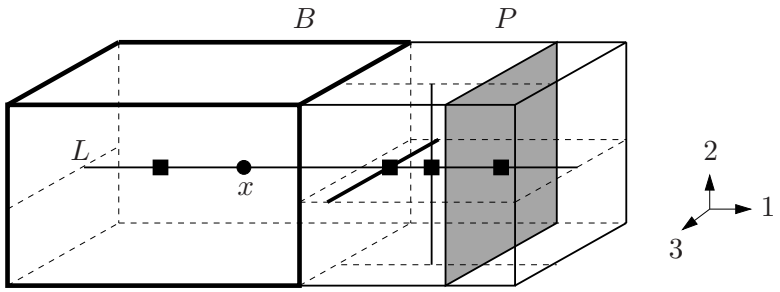

Figure 2.9: Case 2 of Proof of [P1]

We now prove the claim. Let $S = L \cap (P - Refine'(P))$. Since $L \in BadLines'(P)$, $|S| \geq |L \cap P|/8$. In Figure 2.8, the square points indicate the points of S. By induction, for each $y \in S$ there is a cylinder $C(y)$ such that $y \in C(y)$ and $C(y) \cap P \in BadBoxes$. Let $J(y)$ be the set of directions for which $C(y)$ is non-degenerate.

Case 1: There exists a $y \in S$ such that $j \in J(y)$. The box $C(y) \cap B$ has the same cylinder type as $C(y) \cap P$. Therefore, $C(y) \cap B$ is also in $BadBoxes$ and $x \in C(y) \cap B$. In Figure 2.8, the gray plane is $C(y) \cap P$ and contains L. Therefore, $j \in J(y)$. The box $C(y) \cap B$ is just the gray portion inside B and obviously contains x.

Case 2: For all $y \in S$, $j \notin J(y)$. For $J \subseteq [d] - \{j\}$, let $S(J) = \{y \in S : J(y) = J\}$. Refer to Figure 2.9 (for clarity, the part of P extending beyond B to the left is not shown). The direction j is denoted by 1. Therefore, the possible sets for J are $\{2\}, \{3\}, \{2, 3\}$. The three possibilities for $C(y) \cap P$ are lines (in P) along the 2-direction, the 3-direction, or a slice of P spanning directions 2 and 3. This is depicted in the figure for the three (square) points of S.

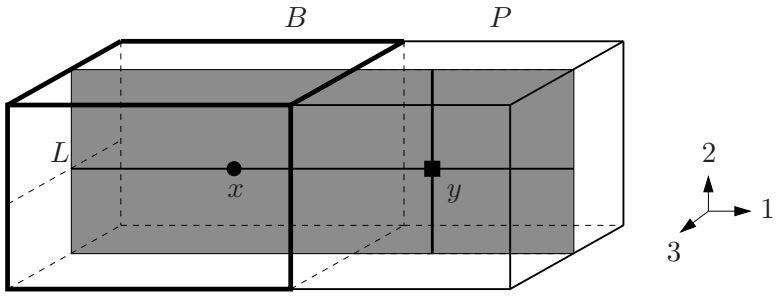

Figure 2.10: Case 2 of Proof of [P1] : Extending to get C

The set J has 2^{d-1} possibilities. By a simple averaging argument, there must be a $J^* \subseteq [d] - \{j\}$ such that $|S(J^*)| \geq |S|/2^{d-1}$ As mentioned before, since $L \in BadLines'(P)$, $|S| \geq |L \cap P|/8$. We get that $|S(J^*)| \geq |L \cap P|/2^{d+2}$. Extending $C(y)$ in direction j gives the same cylinder C for all $y \in S(J^*)$. We then have $P \cap C \in BadBoxes$ since for $1/2^{d+2}$ of $r \in P_j$, $P \cap C \cap H_j(r) \in BadBoxes$. Note that this is why we have the factor of 2^{d+2} in the definition of $BadBoxes$. Refer to Figure 2.10. Suppose that J^* is $\{2\}$, as shown in the figure. We extend $C(y)$ in direction 1 to get C, so the intersection $P \cap C$ is the gray plane shown. If J^* is $\{2, 3\}$, then $P \cap C$ would be the whole of P.

Obviously, $B \cap C$ has the same cylinder type as $P \cap C$ and so is also in $BadBoxes$. This completes the proof of the claim.

Proof of [P2] : We prove by induction on $j = dim(B)$ that if $B \in BadBoxes$ then $B \subseteq \Upsilon^j_\theta(S)$.

If $dim(B) = 1$ then B is a (one-dimensional) interval. $B \in BadBoxes$ implies that at least $1/2^{d+2}$ of the points of B are rejected by *Sift* and so $B \subseteq \Upsilon^1_\theta(S)$.

Now suppose $j \geq 2$. Since $B \in BadBoxes$ there is a non-degenerate direction i of B such that for at least at least $1/2^{d+2}$ of the values $r \in B_i$, $H_i(r) \cap B \in BadBoxes$. Each of the boxes $H_i(r) \cap B$ has dimension $j - 1$, so by induction is a subset of $\Upsilon^{j-1}_\theta(S)$. Therefore for each i-line L that meets B, at least a $\theta = 1/2^{d+2}$ fraction of the points of $L \cap B$ is in $\Upsilon^{j-1}_\theta(S)$, which implies $L \cap B \subseteq \Upsilon^j_\theta(S)$ for all L in direction i meeting B, which implies $B \subseteq \Upsilon^j_\theta(S)$. This completes the proof of [P2].

Chapter 3

Convexity Reconstruction

This chapter deals with distributed filters for convexity in two and three dimensions. For our main result, the input is a three-dimensional terrain (surface generated by a planar graph in the xy-plane with a z-coordinate for each vertex). The queries are made for faces, and the filter outputs the coordinates of the vertices of the input face. The filter guarantees that the terrain it outputs is convex and modifies the minimum number of faces (upto a constant factor) to do so.

There has been work on studying geometric properties from a sublinear perspective [28, 32, 36, 63], but we do not really borrow any techniques from here. A major feature of this result is the variety of geometric tools that are used. Also, we devise many useful techniques to study convexity in sublinear time. One of the most interesting problems explored relates to the classical problem of finding small vertex separators in planar graphs. Given access to a geometric, straight-line embedding of a planar graph, can we hope to find small separators in sublinear time? We provide some partial answers to this problem, but it is by no means the last word.

We first describe a filter for two-dimensional convexity, where the input is a polygon and the queries are made for edges. This is much simpler and provides a warmup for the harder three-dimensional case. We also present an optimal property tester for two-dimensional convexity. We prove lower bounds showing a complexity gap between testing and reconstruction for two-dimensional convexity. At the end, we also provide a lower bound showing why reconstruction for terrains is fundamentally harder than for polygons.

3.1 Convexity filters for polygons

In this section, we shall deal with the problem of reconstructing convexity for polygons. For simplicity, we shall begin with reconstructing *polygonal chains* instead of polygons. For ease of presentation, we will make the assumption that these chains are terrains (the projections of the edges on the x-axis do not overlap). The filter for chains will then be extended to handle general polygons. The input is a 2D polygonal chain $\mathcal{D}$, ie, a polygonal curve with points $p_1, \ldots, p_n$, where each consecutive set of points is joined by a directed line segment. These segments are the *edges* of

$\mathcal{D}$. The set of edges will be denoted by E. The chain $\mathcal{P}$ induces a natural ordering on the edges - given two edges $e = p_i p_{i+1}, f = p_j p_{j+1}$, $e \preceq f$ if $i \leq j$ ($e \prec f$ if $i < j$). We can define the interval $[e, f] = \{g \in E | e \preceq g \preceq f\}$.

The edge set E is stored in an ordered doubly-linked list, from which we can access a random edge and walk from it in either direction. In addition, there exists an oracle that give the ordering of edges (given edges e, f, g, the oracle outputs their order in $\mathcal{D}$). Note that for the case of terrains, this is trivial to implement.

Let $\varepsilon_{\mathcal{D}} n$ denote the minimum number of edges that need to be modified to make $\mathcal{D}$ upper convex, *without* changing the ordering of E. This modification only refers to changing the actual coordinates of the points of $\mathcal{D}$. Lower convexity means that (after the modification) for any edge e, it must point towards the increasing x direction, and *all* edges must lie to the left halfspace defined by e. Therefore, two edges are in convex position with each other if they both point from left to right and lie to the left of each other.

What makes the 2D case remarkable is that a certain easily testable property allows us to classify any given edge in one of two categories (good or bad) in a way that leads to a filtering mechanism with a constant approximation factor. (A similar classification was used for filtering monotone functions in [6].)

Definition 3.1.1. *A pair $\langle e, f \rangle$ is a* violation *if e and f are not in convex position. We also say that e violates f and vice versa.*

The following transitivity relation is immediate: if $e \prec f \prec g$ and $\langle e, g \rangle$ is a violation, then so is at least one of $\langle e, f \rangle$ or $\langle f, g \rangle$. This is a simple consequence of the properties of convexity. Note that for an edge e that points from right to left, the pair $\langle e, f \rangle$ is a violation for all $f \in E$.

Definition 3.1.2. *Given any $0 < \delta < 1/2$, an edge e is called* δ-bad *if there exists an edge f such that either (i) $e \prec f$ and the number of $g \in [e, f]$ that violate e is at least $(1/2 - \delta)|[e, f]|$ or (ii) $f \prec e$ and the number of $g \in [f, e]$ that violate e is at least $(1/2 - \delta)|[f, e]|$. The edge f is referred to as a* witness *to e's badness. An edge that is not δ-bad is called* δ-good.

The next lemma is crucial for proving the correctness of the filter.

Lemma 3.1.3. *(i) The 0-good edges have no violating pairs; (ii) at least $\varepsilon_{\mathcal{D}} n$ edges are 0-bad; and (iii) no more than $(3 + 8\delta/(1 - 2\delta))\varepsilon_{\mathcal{D}} n$ edges are δ-bad.*

Proof. (from [5]) Note that by transitivity, for any $e \prec f$ such that $\langle e, f \rangle$ is a violating pair, either e or f (or both) is 0-bad. Therefore, if we were to remove all the 0-bad edges, the remaining edges would be in convex position; hence (i) and (ii).

We start by assigning to each δ-bad e a witness f_e to its badness (if many witnesses exist, we just choose any one). If $f_e \succ e$, then e is called *right-bad*; else it is *left-bad*. (Obviously, the classification depends on the choice of witnesses.)

Let C be a set of $\varepsilon_{\mathcal{D}} n$ edges where $\mathcal{D}$ can be modified to make it convex. To bound the number of right-bad edges, we charge C with a credit scheme. (Then we apply a similar procedure to bound the number of left-bad edges.) Initially, each

element of C is assigned one unit of credit. For each right-bad $e \notin C$ (in reverse order from right to left), *spread* one credit among all the g such that $e \preceq g \preceq f_e$ and $\langle e, g\rangle$ is a violation (note that g must be in C). We use the word "spread" because we do not simply drop one unit of credit into one account. Rather, viewing the accounts as buckets and credit as water, we pour one unit of water one infinitesimal drop at a time, always pouring the next drop into the least filled bucket.

We now show that no edge in C ever receives an excess of $2 + 4\delta/(1 - 2\delta)$ units of credit. Suppose by contradiction that this were the case. Let e be the right-bad that causes some edge g's (g belongs to C) account to reach over $2+4\delta/(1-2\delta)$. By construction e is not in C; therefore, the excess occurs while right-bad e is charging an edge g such that $e \prec g \preceq f_e$ and $\langle e, g\rangle$ is a violation. Note that, because $e \notin C$, any g satisfying these two conditions belongs to C (let us denote the number of such edges by l) and thus gets charged. With the uniform charging scheme, this ensures that all of these l elements of C have the same amount of credit by the time they reach the excess value, which gives a total greater than $l(2 + 4\delta/(1 - 2\delta))$. By definition of right-badness, $l \geq (1/2 - \delta)|[e, f_e]|$. But none of these accounts could be charged before step f_e; therefore,

$$(1/2 - \delta)|[e, f_e]|(2 + 4\delta/(1 - 2\delta)) < |[e, f_e]|,$$

which is a contradiction.

Of the total of at most $2 + 4\delta/(1 - 2\delta)\varepsilon_{\mathcal{D}}n$ units of credit, $\varepsilon_{\mathcal{D}}n$ units of credit came from initially assigning the edges of C one unit of credit each. Therefore, there are at most $1+4\delta/(1-2\delta)\varepsilon_{\mathcal{D}}n$ right-bad edges. By applying a similar argument for left-bad edges (this time charging from left to right), we prove (iii). □

With this classification of edges, we now give an outline of the filter to convexity. Our aim is to create a *skeleton* of δ-good edges that is sublinear, yet represents the convex properties of $\mathcal{D}$ quite accurately. Then convexification will be done locally ensuring that we create no violations with the skeleton. This will ensure convexity and that at most $O(\varepsilon_{\mathcal{D}}n)$ edges will be modified.

We first describe an offline algorithm *offline*(L), that given an ordered list of edges L, finds a close convex polygon to L.

Claim 3.1.4. *Let L be an ordered list of edges. The procedure* offline(L) *outputs a convex polygon. Furthermore, it generates a matching amongst all edges that are modified such that if $\langle e, f\rangle$ is a matched pair, then $\langle e, f\rangle$ is a violation.*

Proof. The stack S always contains a list of edges in convex position. This is a consequence of transitivity. If e is not in convex position with some edge of S, then it cannot be convex position with the head of S. Whenever edge e is not added to S, then it is matched with the head of S (which is removed from S). □

Next, we design a prcoedure called *skeleton* which constructs a sublinear sized approximate convex structure that "represents" the closest convex polygon to $\mathcal{D}$.

offline (L)

```
let S be an empty stack
for edge e in L (going in order)
    if e is a violation with the head of S
      pop head of S
    else push e into S
keep edges in S as they are, and for all other edges
  linearly interpolate (between edges of S)
  to change their coordinates
```

Figure 3.1: Convexifying offline

skeleton $(\mathcal{D})$

```
choose a random set R of n^{2/3} edges (put them in ordered array)
choose a random set S of n^{1/3} edges
for each edge e ∈ S
  walk along D in increasing order from e for n^{1/3} log n edges
    (call this stretch of edges T)
  if for any f ∈ T, the interval [e,f] has more than a
    (1/2 − δ)-fraction of violations with e, remove e from S
  for all intervals [e,f] such that
    |[e,f] ∩ R| = (1+δ)^j (for some integer j) and |[e,f] ∩ R| > log n
      if [e,f] ∩ R contains more than
      a (1/2 − 4δ)-fraction of violations with e,
        remove e from S
  repeat the above for intervals of the form [f,e]
output S in order
```

Figure 3.2: The procedure *skeleton*

Claim 3.1.5. *The procedure* skeleton *runs in $\widetilde{O}(n^{2/3})$ time. With high probability, it outputs a list of δ-good edges such that - for any interval I of edges of size at least $n^{2/3}\log n$, if I contains at least a δ-fraction of 8δ-good edges then the output contains at least one such edge.*

Proof. First, we show that if an edge e of the random set S is not removed, then it is δ-good. Suppose e is δ-bad. Then there exists some interval $[e, f]$ (or $[f, e]$[1]) such that it has more than a $(1/2 - \delta)$-fraction of violations with f. Suppose $|[e, f]| \leq n^{1/3}\log n$. Then e is definitely removed. If not, then with high probability (by a Chernoff bound argument), the fraction of violations in $R \cap [e, f]$ is at least $(1/2 - 3\delta)$. Therefore, there exists some f' such that $|[e, f'] \cap R| = (1 + \delta)^j > \log n$ and $|[e, f'] \cap R|$ contains more than a $(1/2 - 4\delta)$-fraction of violations with e.

Suppose the interval I contains a δ-fraction of 8δ-good edges. By a Chernoff bound, with high probability, the random set S contains at least one such edge g. Using an argument almost identical to that above, we can show that with high probability, g will not be removed. Taking a union bound over all the error probabilities (there are at most polynomially many events, and the error probabilities are polynomially small), we complete the proof. □

We now get the to the actual reconstruction procedure, *convexify* $(\mathcal{D}, e, \rho)$. The random seed ρ is of length $\widetilde{O}(n^{2/3})$ and is only used to call *skeleton*. In other words, there is one fixed skeleton that all calls to *convexify* use.

Lemma 3.1.6. *The reconstruction procedure* convexify *takes $\widetilde{O}(n^{2/3})$ time and outputs a convex polygon $\mathcal{D}'$ that differs from $\mathcal{D}$ at $(5 + O(\delta))\varepsilon_{\mathcal{D}} n$ edges.*

Proof. The running time is easy to see. We need to bound the number of edges that *convexify* modifies. Consider two consecutive (by the edge ordering) edges f, g in S. Suppose $[f, g] < n^{2/3}\log n$. If an edge $e \in [f, g]$ is not in convex position with either f or g, then, by Claim 3.1.5, it must be δ-bad. Suppose that edge $e \in [f, g]$ is modified by the call to *offline*(L). By Claim 3.1.4, all edges removed by *offline* (over all calls for different interval $[f, g]$) consist of matched pairs. Let C be a set of $\varepsilon_{\mathcal{D}} n$ edges whose removal leaves $\mathcal{D}$ convex. At least one edge in each matched pair belongs to C, since the matched pairs are violations. Therefore, at most $2\varepsilon_{\mathcal{D}} n$ edges can be modified by all the calls to *offline*.

Suppose, on the other hand, that $[f, g] > n^{2/3}\log n$. All the edges in this interval are modified by *convexify*. By Claim 3.1.5, the fraction of 8δ-good edges in $[f, g]$ is atmost δ. Using the bound of part (iii) of Lemma 3.1.3, the total number of modified edges is $(3 + O(\delta))\varepsilon_D n + 2\varepsilon_D n = (5 + O(\delta))\varepsilon_D n$. □

3.1.1 Testing

First, we describe a convexity tester. The input polygon $\mathcal{D}$ is given as in linked list format with a ordering oracle. A tester is a randomized algorithm which, given $\mathcal{D}$ and $0 < \varepsilon < 1$, will do the following in sublinear time :

[1]Since the proof is identical in both case, we will just assume that it is $[e, f]$

convexify $(\mathcal{D}, e, \rho)$

use ρ and the procedure *skeleton* to output the skeleton S
walk both in increasing and decreasing order for $n^{2/3} \log n$ steps
find closest skeleton edges f, g smaller and larger than e
if $[f, g] < n^{2/3} \log n$
    let L be the list of edges between the skeleton edges that
      are in convex position with them (skeleton edges)
    call *offline*(L) to output set of convex edges
    linearly interpolate to get coordinates of all other edges,
      and output e accordingly
else linearly interpolate between f and g
  (changing all edges in between) to output e

Figure 3.3: Performing reconstruction for e

- If $\mathcal{D}$ is convex, output "convex" with prob $> 2/3$
- If $\mathcal{D}$ is ε-far from being convex, output "not convex" with prob $> 2/3$

Our aim is to prove the following theorem.

Theorem 3.1.7. *There exists a tester for convexity of polygons given in linked list format (and with ordering oracle present) using $O(\varepsilon^{-1} n^{1/3})$ lookups and takes $\widetilde{O}(\varepsilon^{-1} n^{1/3})$ time.*

We will analyse the procedure *is-convex* and show that it satisfied the requirements of Theorem 3.1.7. If $\mathcal{D}$ is convex, then the tester obviously accepts with probability 1. From now on, we assume that $\mathcal{D}$ is ε-far from being convex.

Consider a set of εn edges whose removal makes $\mathcal{D}$ convex. The set of edges is denoted by E. Let these edges be called *unsound* edges (this set will be denoted by U, $|U| \geq \varepsilon_{\mathcal{D}} n$). The remaining edges are *sound* (this set will be S). Note that all of S is in convex position and that there cannot be a set of convex edges whose size is larger than $|S|$.

Lemma 3.1.8. *There exists a one-to-one function $\alpha : U \to E$ such that $\forall u \in U$, $\alpha(u)$ violates u.*

Proof. We will construct α through an iterative process. Initially, we start with the set $T = U$. If there exists a violating pair $\langle u_1, u_2 \rangle$ $(u_1, u_2 \in T)$, we assign $\alpha(u_1) = u_2$ and $\alpha(u_2) = u_1$. Then we remove u_1, u_2 from T and repeat. In the end, we end up with a set T of unsound edges which do not violate each other. In other words, T is in convex position. Let us now construct a bipartite graph G with T on one side

is-convex $(\mathcal{D}, \varepsilon)$

choose $c\varepsilon^{-1/2}n^{1/3}$ edges at random (for large enough constant c)
if they are not in convex position, output "not convex"
repeat $c'\varepsilon^{-1}$ times (for large enough constant c')
    choose random edge e and walk in $\mathcal{D}$ for
      $n^{1/3}$ steps forwards and backwards
    if these edges are not in convex position, output "not convex"
output "convex"

Figure 3.4: Testing if polygon $\mathcal{D}$ is convex.

and S on other. Any two violating edges are connected. Take any set $T' \subseteq T$. T' is in convex position and is totally unsound. Therefore, T' should have at least $|T'|$ violators in S. If not, we could replace all these violators in S by T', and we would have a set of convex edges whose size is greater than $|S|$. The number of neighbours of T' in G is $\geq |T'|$. By Hall's Theorem (Theorem 2.1.2 in [38]), G has a perfect matching. For every $u \in T$, we set $\alpha(u)$ to be the edge in S that matches u. This completes the construction of α. □

A contiguous set of unsound edges in convex position will be called a *stretch*.

Claim 3.1.9. *For any stretch L of length k, there exist sets of edges A_L, B_L such that $A_L \subseteq L$, $B_L \cap L = \phi$, $|A_L|, |B_L| \geq k/4$, and every edge in A_L violates every edge in B_L.*

Proof. For every u in the middle $k/2$ edges of L, consider $\alpha(u)$. Without loss of generality, we can assume that at least half of these edges lie in front (along $\mathcal{D}$) of L. Let all these edges be B_L, and let the rightmost $k/4$ edges of L be A_L. By transitivity, every edge in A_L violates every edge in B_L. □

Proof. (Theorem 3.1.7) The lookup complexity bound is obvious. The time complexity of checking if $c\varepsilon^{-1/2}n^{1/3}$ edges are in convex position requires $\widetilde{O}(\varepsilon^{-1/2}n^{1/3})$ time.

We need to show that with high probability, a violation will be detected. The tester has two stages - the first involves sampling $O(n^{1/3})$ edges, and the second involves walks of length $n^{1/3}$. Consider a stretch L of length k. Let $\mathcal{E}_L$ denote the event that some element of A_L is chosen in the first $32\varepsilon^{-1/2}n^{1/3}$ samples and that some element of B_L is chosen in the next $32\varepsilon^{-1/2}n^{1/3}$ samples in the first stage. The probability that $\mathcal{E}_L$ occurs is -

$$\left[1 - \left(1 - \frac{k}{4n}\right)^{32\varepsilon^{-1/2}n^{1/3}}\right]^2 > \left[1 - e^{-8k\varepsilon^{-1/2}n^{-2/3}}\right]^2$$

Case 1 : There exists a stretch L of length $k > \delta\varepsilon^{1/2}n^{2/3}$ (for some small enough constant $\delta > 0$). Then the probability that $\mathcal{E}_L$ occurs is a constant. Choosing c large enough will ensure that $\mathcal{E}_L$ occurs (and that a violation is detected) with probability $> 2/3$.

Case 2 : At least $\varepsilon n/2$ unsound edges lie in stretches of length $< n^{1/3}$. Such an edge will be found with high probability in the second stage. Walking for $n^{1/3}$ steps in both directions along $\mathcal{D}$ ensures that a violation will be detected.

Case 3 : At least $\varepsilon n/2$ unsound edges lie in stretches of length $\geq n^{1/3}$ and all stretches have length $\leq \delta\varepsilon^{1/2}n^{2/3}$. With every stretch L of size $k_L \geq n^{1/3}$ associate a random variable X_L, the indicator variable for the event $\mathcal{E}_L$.

$$\mathbf{E}[\sum_L X_L] > \sum_L \left[1 - e^{-8k_L\varepsilon^{-1/2}n^{-2/3}}\right]^2 \geq \frac{\varepsilon n}{2n^{1/3}} \frac{64n^{2/3}}{4\varepsilon n^{4/3}} \geq 8$$

(Since no stretch has length $> \delta\varepsilon^{1/2}n^{2/3}$ and $x > 0$, we can use the inequality $e^{-x} < 1 - x/2$. A standard convexity argument then gives us the above bound.) Lemma 3.1.8 implies that the X_L's are "almost" pairwise independent. We can show that $var(\sum X_L) \leq 4\delta\mathbf{E}[\sum X_L]$. By Chebyschev's inequality, $\sum X_L$ will take a positive value (and therefore a violation will be detected) with some constant probability. The constant c can be chosen large enough to make the probability of success $> 2/3$. □

3.1.2 General polygons

So far, we made the simplifying assumption that $\mathcal{D}$ looks like the graph of a function in one variable. We now show how to remove this assumption, for both testing and reconstruction. The algorithms given will remain the same - only the correctness proofs need to be changed. We choose some arbitrary edge (referred to as e_0) and orient the axes such that e_0 is parallel to the y-axis and points in the negative direction.

We now modify the definition of a violation. First, we call an edge *forward* if it points in the positive x direction (otherwise, it is called *backward*). Given edges e_1, e_2, the pair (e_1, e_2) is a violation if any of the following hold -

1. e_1 and e_2 are not in convex position.
2. e_1 and e_2 are forward edges, they are in the cyclic order e_0, e_1, e_2 and the x-projection of e_2 is not strictly in front (along the x-axis) of the x-projection of e_1.
3. e_1 and e_2 are backward edges, they are in the cyclic order e_0, e_1, e_2 and the x-projection of e_1 is not strictly in front (along the x-axis) of the x-projection of e_2.
4. e_1 is a backward edge, e_2 is a forward edge, and they are in the cyclic order e_0, e_1, e_2.

We can now modify the proof of Claim 3.1.9. (Note that the proof of Theorem 3.1.7 will then hold for general polygons.)

Claim 3.1.9. For any u in the middle $k/2$ edges of L, consider $\alpha(u)$. Note that if the cyclic order is $e_0, u, \alpha(u)$, then the cyclic order is $e_0, v, \alpha(u)$ for any $v \in L$. (A similar statement can be made if the order is $e_0, \alpha(u), u$.) Using this (and the new definition of a violation), it can be shown that $\alpha(u)$ violates either the first $k/4$ or the last $k/4$ edges. Wlog, we can assume that half of the $\alpha(u)$'s (let this be B_L) violate the first $k/4$ edges of L (let this be A_L). □

Proving that reconstruction still works is relatively easy. Essentially, we need to prove that Lemma 3.1.3 is still correct. The proof of (i), (ii) only uses the transitivity property of convexity. Given edges e_1, e_2, the transitivity property now holds in either $[e_1, e_2]$ or $[e_2, e_1]$. In the proof of (iii), we perform a charging procedure that starts from the rightmost end of $\mathcal{D}$. Now, since $\mathcal{D}$ now has no "rightmost" end, we need to choose some place to start the charging. We can find some edge e' that is surrounded by many good edges. The charging argument used to prove (iii) can now be done by starting from e' (and moving in cyclic and reverse cyclic orders) without affecting the proof.

3.1.3 Lower bounds

We first show the optimality of the convexity tester (upto polylogarithmic factors in n).

Theorem 3.1.10. *Testing convexity in* 2 *dimensions (where the input polygon is given as a linked list) requires* $\Omega(\varepsilon^{-1/2}n^{1/3})$ *lookups.*

Proof. We will give a lower bound for the following problem : given an input polygon in linked list format which is ε-far from being convex, output a violation. We use Yao's minimax principle. We will consider a distribution on the input, and prove a lower bound on any deterministic algorithm that gives the correct answer with probability 2/3 over the input.

Suppose some deterministic algorithm makes $\delta\varepsilon^{-1/2}n^{1/3}$ lookups(δ is a sufficiently small constant). The input is a linked list of all the edges. In our model (taken from [28]), the list is accessed through a table $T[1 \cdots n]$, where the i-th element is stored in location $\sigma(i)$ - so $T[\sigma(i)] = i$. Consider the following polygon - $\mathcal{D}$ has $\varepsilon n^{2/3}$ stretches of length $n^{1/3}$. Each stretch violates $3n^{1/3}$ edges. Figure 3.5 shows a portion of $\mathcal{D}$ with the stretches in bold. $\mathcal{D}$ is constructed by stitching together many copies of this structure. The stretches occur far apart and therefore do not violate one another.

The input distribution is formed by choosing the permutation σ uniformly at random from the symmetric group on n elements. Any deterministic algorithm can be seen as executing a sequence of steps of the form : (A) choose a location $T(l)$ and look up $T(\sigma(i \pm 1))$, where $l = \sigma(i)$ (this is equivalent to walking along the

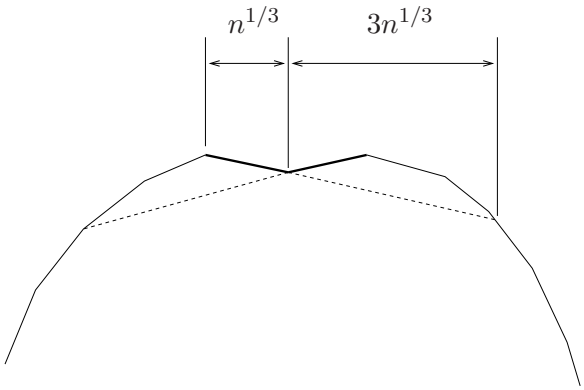

Figure 3.5: Testing Lower Bound

polygon); (B) compute a new index l based on previous indices and look up $T(l)$. We can see that $\sigma^{-1}(l)$ is equally likely to lie anywhere in the unvisited portion of $\mathcal{D}$.

For every stretch L, let X_L be the indicator variable for the event that a violation from L is detected by 2 B-steps. X_L is 1 iff an edge from L and an edge from the violations of L (having size $3n^{1/3}$) are chosen.

$$\mathbf{E}[X_L] < \left[1 - \left(1 - \frac{4n^{1/3}}{n}\right)^{\delta\varepsilon^{-1/2}n^{1/3}}\right]^2 < \left[1 - e^{-8\delta\varepsilon^{1/2}n^{-1/3}}\right]^2 < \frac{64\delta^2}{\varepsilon n^{2/3}}$$

$$\mathbf{E}[\sum_L X_L] \leq \varepsilon n^{2/3}\mathbf{E}[X_L] < 64\delta^2$$

By Markov (and choosing a small enough δ), the probability that $\sum X_L$ exceeds 1 is less than 1/10. In the following, c_1 and c_2 denote some sufficiently large constant. Let Y_L be the indicator variable for the event that a B-step falls within $\varepsilon^{-1/2}n^{1/3}/c_1$ of either boundary of L.

$$\mathbf{E}[Y_L] < 1 - \left(1 - \frac{4n^{1/3}}{c_1\sqrt{\varepsilon}n}\right)^{\delta\varepsilon^{-1/2}n^{1/3}} < \frac{\delta/c_2}{\varepsilon n^{1/3}}$$

$$\mathbf{E}[\sum_L Y_L] \leq \varepsilon n^{2/3}\mathbf{E}[Y_L] < (1/c_2)\delta n^{1/3}$$

Again by Markov, the probability that $\sum Y_L$ exceeds $(1/10)\delta n^{1/3}$ is less than 1/10. With probability at least 8/10, B-steps by themselves did not detect a violation, and at most a 1/10 fraction of edges visited by B-steps fall within $\varepsilon^{-1/2}n^{1/3}/c_1$ of a stretch boundary. Assume that this happens. Any violation detected must involve an A-step. With probability at most 1/10, $\leq \varepsilon^{-1/2}n^{1/3}/c_1$ A-steps were

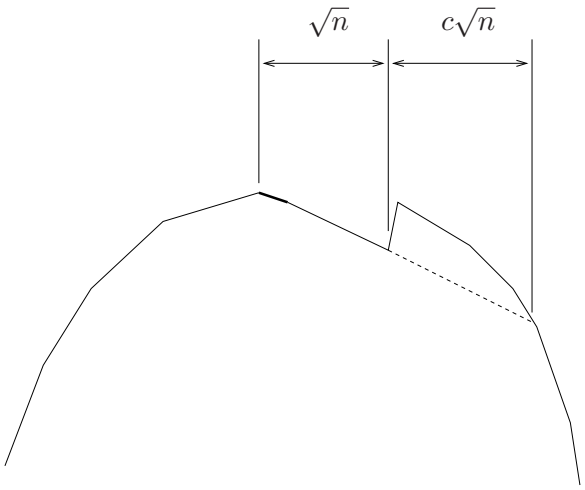

Figure 3.6: Reconstruction Lower Bound

needed for detection. Since the total number of steps is $\delta\varepsilon^{-1/2}n^{1/3} < \varepsilon^{-1/2}n^{1/3}/c_1$, (a union bound shows that) the algorithm will fail with probability $> 7/10$. □

We now show a $O(\sqrt{n})$ lower bound[2] for reconstructing convexity. This, combined with the with the $\widetilde{O}(n^{1/3})$ algorithm for testing shows a complexity gap (in terms of the input size n) between testing and reconstruction.

Theorem 3.1.11. *Any filter for 2-dimensional convexity requires $\Omega(\sqrt{n})$ lookups per query, where the input is given in linked list format.*

Proof. We prove a lower bound for the following problem - given $\mathcal{D}$ and an edge e, output whether e is good or bad. If e is deemed bad, then a violating edge must be provided. All good edges must be in convex position, and the number of edges deemed bad must be $< c\varepsilon_{\mathcal{D}}n$ (for some constant c). We use Yao's minimax principle and proceed exactly as in the proof of Lemma 3.1.10. We prove a lower bound for a deterministic algorithm that gives the correct answer on a distribution of inputs with probability $> 2/3$. Consider a polygon that has distance εn (for some $\varepsilon > 0$) from convexity that has stretches of violating edges as shown in Figure 3.6. Essentially, the polygon is constructed by pasting together many stretches as given in the figure, and ensuring that the stretches do not violate each other. The downward pointing part that comes after the stretch of $\sqrt{n}$ is just one edge. It is obvious that the bold edge e is bad. The problem here is to find some edge in the $c\sqrt{n}$ range.

The distribution is formed by choosing the permutation σ at random (in the same model as that in the proof of Lemma 3.1.10). We are given a query for the edge e.

[2]This lower bound holds even for sequential reconstruction.

Using an argument similar to the one for the testing lower bound, we can show that at least $\delta\sqrt{n}$ (for some constant δ) lookups are required to detect a violation.

□

3.2 A convexity filter for 3D terrains

The dataset is an n-face triangulated terrain $\mathcal{D}$ represented in standard DCEL fashion. We assume that the xy-projection of any face of $\mathcal{D}$ is a triangle with both edge lengths and angles bounded above and below by constants (bounded aspect ratio condition). The reconstructed terrain $\mathcal{D}^c$ is convex in the sense of being the boundary of the upper hull of its vertex set. There are various equally reasonable definitions of the parameter $\varepsilon_{\mathcal{D}}$. For simplicity, we define $\varepsilon_{\mathcal{D}} n$ as the minimum number of faces of $\mathcal{D}$ that need to be removed in order to make the terrain convex. Note that this definition does not require us to "patch the holes." Choosing to do so, however, would only increase the distance by a constant factor, which, for the purpose of our filter, is immaterial. The edge table allows us to sample random edges. From this, it is elementary to implement a uniform sampler for triangular faces as well. (To sample vertices would be more difficult but, fortunately, we do not need that feature.)

The idea is to break up the terrain into connected *patches* of suitable size by removing a small set $\mathcal{F}$ of separating faces. The *fence* $\mathcal{F}$ decomposes the terrain into connected patches of suitable sizes. A critical feature of $\mathcal{F}$ is to be of sublinear size. To achieve this, we use a sublinear version of the classical planar separator theorem. The weakening is required to make the computation sublinear. The final processing step is to convexify $\mathcal{F}$. This is a delicate operation which cannot be performed in isolation with the rest of the terrain: this is a perfect illustration of why early decisions are crucial in online filtering.

We define a suitable range space (of unbounded VC dimension!) whose sampling gives us enough global information about the whole terrain to guide the reconstruction of $\mathcal{F}$. The convexified $\mathcal{F}$ fences off the patches in such a way that it is possible from then on to answer any subsequent query by treating its associated patch in isolation from the rest of the terrain. But, before we can get to online reconstruction, we need to define two key procedures: one is an offline algorithm for convexifying the terrain within twice the minimum distance; the other estimates the distance $\varepsilon_{\mathcal{D}}$ in sublinear time.

Any filter must explore both global and local properties. The difficulty lies in gathering enough information in sublinear time. Any approach must involve a combination of sampling and local search. The filter essentially works as follows: first, it estimates $\varepsilon_{\mathcal{D}}$ using the sublinear time procedure. If the distance is very small, then the offline algorithm is used for convexification. Otherwise, it constructs a fence $\mathcal{F}$ and convexifies it. It is critical that this convexification be done by taking the global structure into account—this is achieved by choosing a large enough sample of faces of $\mathcal{D}$ (which then is used to define the range space mentioned in the previous paragraph) and convexifing $\mathcal{F}$ so that it is in convex position with most of the

OFFLINE-RECONSTRUCTION

```
Initialization:  T ← ∅
  for each face f of D:
    if f is in convex position with T
      then T ← T ∪ {f}
      else  find face g ∈ T not in
          convex position with f
           T ← T \ {g}
    output D^c ← T
```

sample. This creates a "skeleton" which captures the global properties of $\mathcal{D}$ and also splits $\mathcal{D}$ into a set of small connected patches. Now, each patch is reconstructed independently, ensuring that it stays in convex position with the convexified $\mathcal{F}$. Since a patch is a small connected portion of $\mathcal{D}$, it can be visited exhaustively by local search (thereby, the filter gains information about the local properties of $\mathcal{D}$). In the following subsections, we discuss the various components that constitute the filter. Finally, we put the pieces together and describe the filter itself.

3.2.1 Offline reconstruction

We describe a 2-approximation offline convexification algorithm, ie, one that, given $\mathcal{D}$ as input, returns a terrain $\mathcal{D}^c$ that is convex and is at distance at most $2\varepsilon_{\mathcal{D}} n$ from it. Note that $\mathcal{D}^c$ can have holes. The convexification proceeds incrementally. Beginning with the empty terrain T, we consider each face of $\mathcal{D}$ one by one and add it to T if it is in convex position[3] with every face currently in $\mathcal{D}^c$. To do this in quasilinear time, we maintain T in a dynamic data structure which supports insertion, deletion, and queries in amoritized poly-logarithmic time.

Denote by T_v (resp. T_p) the set of vertices (resp. face-supporting planes) in T. A terrain face f is in convex position with T if and only if (i) its three vertices lie below each plane in T_p; and (ii) the points of T_v all lie below the plane supporting f. By duality, both tests can be reduced to *dynamic halfspace emptiness* in 3D: maintain a set of points under insertion and deletion, and for any query plane find whether whether all points lie on one side, and if they do not, report one point on each side. Chan [25] has given a halfspace range reporting algorithm which allows us to do that in $O(\log^6 n)$ amortized time for each query/insert/delete.

Whenever the procedure finds a face f that is not in convex position with T, then a face $g \in T$ that is not in convex position with f is removed. Consider a minimal subset U of $\varepsilon_{\mathcal{D}} n$ faces that need to be removed to make $\mathcal{D}$ convex. One of

[3] Two triangles are said to be in convex position if both of them are faces of their convex hull.

f or g has to be present in U. This ensures that the total number of faces removes is at most $2\varepsilon_{\mathcal{D}} n$.

Theorem 3.2.1. *Offline convex reconstruction of an n-face terrain can be performed with an approximation factor of 2 in $\widetilde{O}(n)$ time.*

3.2.2 Estimating the distance to convexity

Consider the violation graph G whose nodes are the faces of $\mathcal{D}$ and whose edges join any two faces not in convex position. Removing all the faces that correspond to a vertex cover of this graph makes $\mathcal{D}$ convex. The minimum vertex cover is of size $\varepsilon_{\mathcal{D}} n$, and any maximal matching M in G is of size $|M| \leq \varepsilon_{\mathcal{D}} n \leq 2|M|$. The offline reconstruction algorithm essentially finds such a maximal matching in G. Fix any constants $0 < \alpha < \beta < 1$ such that $\alpha \geq \frac{1}{2}(3\beta - 1)$, $\alpha \geq (2\beta - 1)$, and let S be a random sample formed by picking each vertex of G independently with probability $p = n^{\alpha-\beta} \log n$. The offline reconstruction algorithm can be used to build a maximal matching M_S for the subgraph G_S of G induced by S. The sample S can be easily specified in $O(|S|)$ time, so that computing M_S takes $\widetilde{O}(pn)$ time, which is $\widetilde{O}(n^{1+\alpha-\beta})$. As we show below, knowing M_S allows us to distinguish between the two cases: $\varepsilon_{\mathcal{D}} \geq n^{-\alpha}$ and $\varepsilon_{\mathcal{D}} \leq n^{-\beta}$.

1. $\varepsilon_{\mathcal{D}} \geq n^{-\alpha}$: Fix some maximal matching M of G. The size of M is $\geq n^{1-\alpha}/2$. If ξ is the number of edges of M in G_S, then $\mathbf{E}\xi = p^2|M|$. Since M is a matching, ξ can be expressed as the sum of independent random variables. By Chernoff's bound [11], $\text{Prob}\,[\,\xi < \frac{1}{2}p^2|M|\,] < e^{-\Omega(p^2|M|)} = e^{-\Omega(p^2 n^{1-\alpha})} = e^{-\Omega(n^{1+\alpha-2\beta}\log^2 n)}$. Since $\alpha \geq 2\beta - 1$, with high probability $\xi \geq \frac{1}{2}p^2|M|$. This implies that G_S contains a perfect matching of size at least $\frac{1}{2}\,p^2|M| \geq \frac{1}{4}\,p^2 n^{1-\alpha}$. Its minimum vertex cover is at least that size; therefore, $|M_S| \geq \frac{1}{8}\,p^2 n^{1-\alpha}$.

2. $\varepsilon_{\mathcal{D}} \leq n^{-\beta}$: Now, the size of M is $\leq n^{1-\beta}$. If χ denotes the number of vertices of M in S, then $\mathbf{E}\chi = 2p|M|$ and, by Chernoff's bound, $\text{Prob}\,[\,\chi \geq 2p|M| + y\,] < e^{-y^2/|M|}$, for any $y > 0$. Setting $y = \frac{1}{8}\,p^2 n^{1-\alpha} - 2p|M| > \frac{1}{9}\,p^2 n^{1-\alpha}$, we find that $\text{Prob}\,[\,\chi \geq \frac{1}{8}\,p^2 n^{1-\alpha}\,] < e^{-\Omega(p^4 n^{1+\beta-2\alpha})} = e^{-\Omega(n^{1+2\alpha-3\beta}\log^4 n)}$. Since the vertices of M provide a vertex cover for G, it follows that, with high probability, $|M_S| < \frac{1}{8}\,p^2 n^{1-\alpha}$.

Theorem 3.2.2. *Given any small $\delta > 0$ and any constants $0 < \alpha < \beta < 1$ such that $\alpha \geq \frac{1}{2}(3\beta - 1)$ and $\alpha \geq (2\beta - 1)$, we can compute a 0/1 bit $b(\mathcal{D})$ in $\widetilde{O}(n^{1+\alpha-\beta})$ time, such that $b(\mathcal{D}) = 0$ if $\varepsilon_{\mathcal{D}} \geq n^{-\alpha}$, $b(\mathcal{D}) = 1$ if $\varepsilon_{\mathcal{D}} \leq n^{-\beta}$, and $b(\mathcal{D})$ takes on any value otherwise.*

3.2.3 Fencing off the terrain

Here we describe an algorithm that finds a sublinear set of faces (the fence) of $\mathcal{D}$ whose removal breaks $\mathcal{D}$ into connected components (patches) also of sublinear

size. Let G be the planar triangulation formed by the projection of $\mathcal{D}$ onto the xy-plane. Note the because of the bounded aspect ratio condition, any triangle in G has bounded sides and angles. The main step in constructing the fence is to design a sublinear algorithm to find balanced planar separators in such graphs. A *balanced planar separator* for a planar graph G with n vertices is a set of vertices whose removal separates G into connected components, each having size $< cn$ (where c is some constant less than 1). Lipton and Tarjan [57] gave the first linear time algorithm to find a balanced planar separator of size $O(\sqrt{n})$. Henceforth, by planar seperator, we always refers to balanced separators. Our aim is to find a set of *faces* in G to remove (note that upto constant factors, this gives a vertex separator of the same size). We are able to beat the linear time bound because we are provided with a geometric embedding of the graph. We also assume the bounded aspect ratio condition.

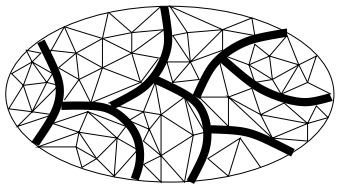

Figure 3.7: The thick black line, the fence, is a itself collection of $o(n)$ triangles.

The algorithm first randomly selects a set of $c\sqrt{n}\log n$ faces, for some sufficiently large constant c. This sample (call it R) is then used to guide the separator. A random starting vertex v in G is chosen, and then R is used to find geometric paths (paths in the plane) from v to the boundary of G which pass through at most $O(\sqrt{n})$ faces. These paths can then be shown to generate a planar separator of $O(\sqrt{n})$ size. We begin by stating the main lemma of this section (this lemma is of independent interest).

Lemma 3.2.3. *For any vertex $v \in G$ (where G is a bounded aspect ratio planar graph with an embedding), there exists a geometric path from v to the boundary of G that passes through $O(\sqrt{n})$ faces. Furthermore, this path can be shown to be x and y-monotone, consisting only of vertical or horizontal line segments. This path can be constructed in $\widetilde{O}(\sqrt{n})$ time.*

We will need to prove some smaller claims before we can attack this lemma. Consider the area defined by a horizontal line segment, and two rays pointing in the positive y-direction. This is called a *vertical slab*. We will only consider line segments that have $\Theta(1)$ length. Now, we assign a charge to each triangle t (face) of G, and *spread* the charge out evenly in the area of t. The total amount of charge throughout G is n. The charge contained in a slab S can be determined by the following fact - for a triangle t, if an f-fraction of it (area-wise) lies inside S, then t contributes f amount of charge.

Claim 3.2.4. *A line segment ℓ of constant size can only intersect a constant number of triangles.*

Proof. First consider two triangles ΔABC and ΔBCD and suppose that ℓ intersects AB, BC, CD. Let ℓ intersect AB at E and BC at F. Refer to Figure 3.8. Wlog, assume that $BF > FC$. Take ΔBEF. If $BE > BF/2$, by the bounded angle condition, we get the EF is at least a constant. On other hand, if $BE \leq BF/2$, then by triangle inequality, we get that $EF \geq BF/2$ and is therefore a constant. The portion of ℓ inside ΔABC and ΔBCD has length $\Theta(1)$. Now take all triangles that intersect ℓ (let us not consider the triangles that contain the endpoints of ℓ; there are only two of them). Take the set of triangles that all share one endpoint and put them in the order in which they intersect ℓ - there can only be a constant number of them, because the angles are bounded. Note that the last two triangles in this sequence have the structure of ΔABC and ΔBCD, where ℓ intersects AB, BC, CD. This shows that there can only be constant number of such sets of triangles, completing the proof of the claim.

□

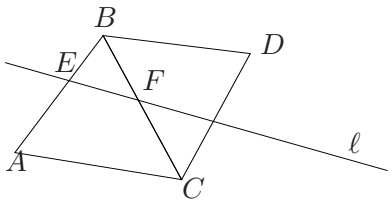

Figure 3.8: Wedge

Claim 3.2.5. *If a slab intersects k triangles, then it contains $\Omega(k)$ amount of charge.*

Proof. Any triangle that is completely inside the slab S contributes one unit charge (all the charge it contains). The main problem is to deal with triangles that intersect the boundary of S. Let us denote by ℓ the line segment (assume its horizontal) defining S, and let r_1 and r_2 be the rays. Abusing notation, ℓ also denotes the length of this segment.

By Claim 3.2.4, only a constant number of triangles can intersect ℓ. We shall now only consider triangles that intersect r_1 or r_2 - if a triangle does not intersect any edge of the boundary of S, then it contributes one unit of charge to S. First consider a triangle t with no vertex inside S. Two edges of t intersect both r_1 and r_2. Because the slab has constant width, wlog, the portion of r_2 between these edges is of constant length. This implies that t contributes $\Omega(1)$ units of charge to S.

Take a triangle t which has a vertex v inside S and whose edges intersect both r_1 and r_2 (call this a triangle of Type 1). Consider the triangle with vertex v and the intersection points of the edge opposite to v with r_1 and r_2. This triangle has constant area and again t contributes $\Omega(1)$ units of charge. Now take a triangle t with

a vertex v inside S which only intersects some r_i (i is either 1 or 2). Furthermore, assume that the perpendicular distance of v from r_i is $\geq \ell/2$. This is a triangle of Type 2. Such a triangle can be easily seen to contribute $\Omega(1)$ units of charge to S. Finally, we take the last case : triangle t has vertex v inside S, its edges only interesect some r_i, and the distance of v to r_i is $< \ell/2$. But, there must be some triangle having v as a vertex which is of type 1 or 2. Since the number of triangles incident to one vertex is $\Theta(1)$, this implies that only a constant fraction of k triangles can be of the final kind. This completes the proof of the claim.

□

We now complete the proof of Lemma 3.2.3.

Proof. We first describe the algorithm used to find such a path from a starting vertex v. A random sample of faces R, of size $c\sqrt{n}\log n$ (where c is a sufficiently large constant) is chosen. It is easy to show that if some slab intersects $> c\log n$ triangles in R, then the slab intersects $> c_1\sqrt{n}$ triangles in G. On the other hand, if the slab intersects $\leq c\log n$ triangles in R, then it intersects $< c_2\sqrt{n}$ triangles in G (c_1, c_2 are some fixed constants, and these hold with high probability).

We draw a constant length horizontal segment ℓ from v in the positive x-direction. By Claim 3.2.4, ℓ intersects at most a constant number of triangles. Suppose the vertical slab defined by ℓ intersects $\leq c\log n$ triangles of R. Then we draw a vertical line (going in the positive y-direction) from ℓ to the boundary (thereby completing the path). If the vertical slab intersects $> c\log n$ triangles of R, then we move in the horizontal direction by drawing another constant length horizontal segment from the right endpoint of ℓ. If the horizontal segments eventually hit the boundary of G, then the path is complete.

The path finally obtained has size $O(\sqrt{n})$. Note that the vertical part of the path can intersect at most $O(\sqrt{n})$ triangles. Each constant length horizontal segment (except for probably the rightmost one) defines a vertical slab intersecting $\Omega(\sqrt{n})$ triangles. By Claim 3.2.5 the amount of charge in this slab is also $\Omega(\sqrt{n})$. All these slabs are disjoint and the total amount of charge overall is n. There can only be $O(\sqrt{n})$ such horizontal segments, and the total number of triangles intersected by these is $O(\sqrt{n})$.

□

This brings us to the final theorem of this section, where we show how a sublinear sized fence which leaves patches of sublinear size can be constructed in sublinear time.

Theorem 3.2.6. *There exists a $\widetilde{O}(\sqrt{n})$ algorithm that finds a balanced planar separator of size $O(\sqrt{n})$ in any triangulated bounded aspect ratio planar graph G which is provided as a DCEL (G is given by straight line embedding) with the planar coordinates of the vertices. For any $0 < a < 1$, this can be used to find a fence of size $O(n^{1-a/2})$ in $\widetilde{O}(n^{1-a/2})$ time such that the patches have size $O(n^a)$.*

Proof. First, we describe how to get a planar separator. Choose a vertex v at random. With probability $\geq 3/5$, the x-coordinate of v is the middle $(3/5)n$ positions in the sorted order of vertices based on x-coordinate. Similarly, this hold for y-coordinates. Therefore, with constant probability, v is the middle $(3/5)n$ positions for both x-sorted and y-sorted lists of vertices. Wlog, there are at least $n/10$ vertices with *both* x and y-coordinates larger than those of v and at least $n/10$ vertices with both x and y-coordinates less than those of v. By choosing $O(\log n)$ vertices uniformly at random, we can ensure (with high probability) that we find at least one vertex that satisfies this property.

Lemma 3.2.3 tells us that there is a geometric path intersecting $O(\sqrt{n})$ triangles that starts from v and ends at the boundary of G. Also, this path (when directed from v) only increases in the x-direction and *decreases* in the y-direction. Similarly, there is a path starting from v that only decreases in the x-direction and increases in the y-direction. These paths can be found in $\widetilde{O}(\sqrt{n})$ time. Together these paths form a separator of G. This is also balanced, since no component can have size larger than $9n/10$.

Recursive application of this procedure yields a fence of size $O(n^{1-a/2})$ wth patches of size $O(n^a)$. The total running time is $O(n^{1-a/2}\log n)$. □

Finding separators in general terrains

As an aside, we describe an algorithm that finds a fence for a general planar graph $\mathcal{G}$ (not nessarily embeddable with bounded aspect ratio). Pick a random sample of $r = n^a$ edges in G, for fixed $0 < a < 1$, and build its (say, x-oriented) trapezoidal map $\mathcal{M}_r$. As is well known, with high probability, each trapezoid intersects $O((n/r)\log n)$ triangles. Consider the dual $\mathcal{H}_r$ graph of $\mathcal{M}_r$, where each node is a trapezoid and two nodes are joined if the corresponding (closed) trapezoids intersect. The graph is planar and so, by iterated application of the planar separator theorem [57], for any fixed $0 < b < 1$, we can find, in $O(r\log r)$ time, a set V of $O(r^{1-b/2})$ nodes whose removal leaves $\mathcal{H}_r$ with no connected component of size exceeding r^b. The fence $\mathcal{F}$ is the set of all the terrain's faces whose projections intersect the trapezoids associated with the nodes of V. With high probability, the fence consists of $O(r^{1-b/2}(n/r)\log n)) = O(n^{1-ab/2}\log n)$ triangles and its removal from the terrain leaves connected patches, each one consisting of $O(n^{1+ab-a}\log n)$ triangles. Note that, because it involves triangles (and not trapezoids), the removal may create much greater fragmentation than is caused within $\mathcal{H}_r$ by the removal of V. Finding the fence takes time $O(n^a\log n + n^{1-ab/2}\log n)$ time, Renaming ab by b, we have proven:

Lemma 3.2.7. *Let G be any planar graph provided with a geometric embedding. For any $0 < b < a < 1$, in $O((n^a + n^{1-b/2})\log n)$ time, it is possible to find a fence $\mathcal{F}$ consisting of $O(n^{1-b/2}\log n)$ triangles, whose removal from the terrain leaves connected patches consisting of $O(n^{1+b-a}\log n)$ triangles each.*

3.2.4 Reconstructing the fence

It might be tempting to convexify the fence by applying the offline algorithm to it, but this could doom the reconstruction of the other faces. Instead, we must allow the global shape of the terrain to influence the reconstruction. To do so, we choose a random sample Σ of the faces of $\mathcal{D}$—the size of Σ shall be set later. Let Σ^c be the offline convexification of the terrain Σ provided by Theorem 3.2.1, and let Σ^f the intersection of the halfspaces bounded above by the planes supporting the faces of Σ^c (the dual convex hull). The reconstructed fence $\mathcal{F}^c$ is obtained by lifting the triangles of $\mathcal{F}$ vertically and "wrapping" them over the surface of Σ^f. Note that such a lifting could replace one fence face by many faces, but based on the bounded aspect ratio condition, we have a bound (proven later in this section) for the total size of $\mathcal{F}^c$. (The bound holds only when Σ is sufficiently large, but our choice of Σ will ensure that.)

Lemma 3.2.8. *The size of $\mathcal{F}^c$ is $\widetilde{O}(n^{1-a/4})$.*

We now explain why $\mathcal{F}^c$ captures the global structure of $\mathcal{D}$. Henceforth, the term "face" refers to a triangle in 3-dimensions (we introduce this because we later use "triangle" to refer to an object in the 2-dimensional plane). We define a range space $(X, \mathcal{R})$, which, although of unbounded VC dimension, has enough sampling power to guide the convexification of the fence. Regarding both $\mathcal{D}$ and $\mathcal{F}$ as sets of faces, we define the ground set $X = \mathcal{D} \setminus \mathcal{F}$. Given two sets S, T of faces, let $\kappa(S, T)$ be the set of faces in T that are not in convex position with at least one face of S. Considering all possible sets Γ of $|\mathcal{F}^c|$ faces, we define $\mathcal{R} = \{\, \kappa(\Gamma, X) \,:\, |\Gamma| = |\mathcal{F}^c| \,\}$.

Let us represent a face f by 12 reals - 9 for the vertices of the face, and 3 for the point in dual space which corresponds to the plane containing f. The face f can be seen as a point $p_f \in \mathbb{R}^{12}$. (Note that the latter 3 reals are completely determined by the former 9 reals. This is a redundant description and actually not necessary for what follows, but it will be helpful.) Consider faces $f \in \Gamma$ and $x \in X$. The convex position of f with respect to x is completely determined by - the position of the vertices of f with respect to the plane containing x, and (in dual space) the position of f, which is a point, with respect to the three planes corresponding to the vertices of x. Let us move to $\mathbb{R}^{12}$. There is an arrangement of 4 hyperplanes in this space, such that the convex position x with respect to f is completely determined by the position of p_f in this arrangement. By taking the necessary planes for all $x \in X$, we get an arrangement of $O(n)$ planes which completely determine the convex position of f with respect to every face in X.

We can view Γ as a point in $p_\Gamma \in \mathbb{R}^{12|\mathcal{F}^c|}$. Using the construction given above for every face in Γ, it is easy to see why the convex position status of each face in Γ with respect to X is completely specified by the location of p_Γ in a certain $12|\mathcal{F}^c|$-dimensional arrangement of $O(n|\mathcal{F}^c|)$ hyperplanes - we have a set of $O(n)$ hyperplanes for every face in Γ. It follows that the primal shatter function φ grows as $\varphi(m) = O(m|\mathcal{F}^c|)^{12|\mathcal{F}^c|}$. With high probability, if Σ is chosen to be a random sample of X of size $O(r^2|\mathcal{F}^c| \log |\mathcal{F}^c|) = \widetilde{O}(r^2 n^{1-a/4})$, it is a $(1/r)$-approximation for $(X, \mathcal{R})$, for any $r > 0$.

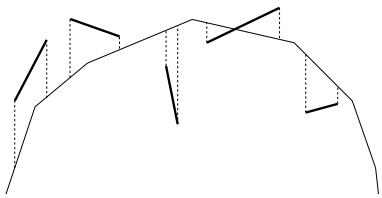

Figure 3.9: The fence is reconstructed by lifting its faces to the upper boundary of Σ^c.

By Theorem 3.2.1, the number of triangles in Σ that were modified during the convexification is at most $2\varepsilon_\Sigma|\Sigma|$. Using an argument similar to the proof of Theorem 3.2.2, we can show that, with high probability, this number is $O(\varepsilon_\mathcal{D}|\Sigma|)$. This implies that $|\kappa(\mathcal{F}^c, \Sigma)| = O(\varepsilon_\mathcal{D}|\Sigma|)$. Since

$$\left| \frac{|\kappa(\mathcal{F}^c, \Sigma)|}{|\Sigma|} - \frac{|\kappa(\mathcal{F}^c, X)|}{|X|} \right| \leq \frac{1}{r},$$

we could easily bound the "damage" caused by the convexification of the fence as follows:

Lemma 3.2.9. $\kappa(\mathcal{F}^c, X) \leq nr^{-1} + O(\varepsilon_\mathcal{D} n)$.

We now prove Lemma 3.2.8. For ease of notation, we shall assume that $\mathcal{F}$ has $O(n^u)$ faces and Σ has size $\widetilde{O}(n^v)$. We will be concerned only with xy-projections of $\mathcal{D}$ and Σ^f. In the following proof, *triangle* refers to the xy-projection of a face of $\mathcal{D}$, while *facet* refers to the xy-projection of a face of Σ^f. *Edges* refer to the edges of triangles. Note that all facets are convex, disjoint from each other (except for their boundaries) and contain a triangle. By the bounded aspect ratio assumption, the radius lengths of the incircle and circumcircle of any triangle are bounded from above and below (by some constant). Let v' be some value less than v.

Definition 3.2.10. *For $\alpha < n^{-(1-v')}$, an* α-thin facet *is a facet with two edges e_1, e_2 such that the minimum distance between e_1 and e_2 is less than α and the angle between e_1 and e_2 is also less than α. The edges e_1 and e_2 are called* sharp *edges. The* min-thinness *of a facet f is the minimum α such that f is α-thin.*

The sharp edges of an α-thin facet form a *wedge* that contains the facet (Figure 3.10).

Claim 3.2.11. *With high probability, there are at most $O(\alpha n)$ α-thin facets.*

Proof. Consider some α-thin facet f which contains triangle t. The distance between t and the sharp edges is at least $\Omega(\alpha^{-1})$ (since t must be inside the wedge created by these sharp edges, and has at least constant in-radius). There exists a rectangle of $\Omega(\alpha^{-1})$ width and $\Omega(1)$ height that is present completely inside f but does not

intersect t (Figure 3.10). Therefore, we can also show that there exist at least $\Omega(\alpha^{-1})$ edges of $\mathcal{D}$ that have at least a constant fraction of their length inside f (let these edges be *bad* for f). Note that the offline convexification must have removed these bad edges. Let the number of α-thin facets be M. Since all facets are disjoint, an edge can be bad for only a constant number of α-thin facets. The total number of bad edges is $\Omega(\alpha^{-1}M)$. With high probability (by taking a Chernoff bound), at least $(c\alpha^{-1}Mn^{v-1}\log n)$ of these edges are chosen in Σ (for some constant c). But this quantity has to be $O(n^v \log n)$, since that is the number of edges removed by convexification. This implies that the number of α-thin facets is $O(\alpha n)$.

□

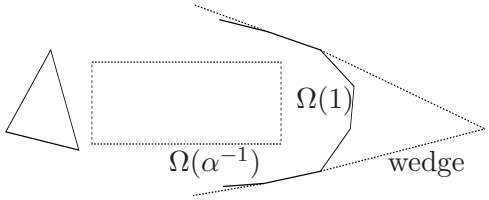

Figure 3.10: Wedge

Claim 3.2.12. *For any $v' < v$, the total complexity of the lifted fence is $O(n^{1+u-v'} + n^{v'} \log n)$.*

Proof. The complexity of lifting a fence face is simply the complexity of the corresponding triangle when laid over the xy-projection of Σ^f. A triangle is said to *generate* all the faces created by overlaying. The total complexity of all triangles generating $O(n^{1-v'})$ faces is $O(n^{1+u-v'})$ (since there are at most $O(n^u)$ fence triangles). Consider some triangle t generating $k > n^{1-v'}$ faces. Each of these faces is created by the intersection of t with a facet.

We will first show that many of these facets are k^{-1}-thin. Since triangles do not intersect, no facet can be completely contained inside t. At least $\Omega(k)$ facets intersect some edge e of t. We take a circle C of constant (but large enough) radius such that C contains e and the minimum distance between e and C is $\Omega(1)$. The intuition for the following proof is quite simple—since C is a constant sized circle and many facets intersect it, many facets have to be thin (Figure 3.11). Since the area of a facet is $\Omega(1)$, there can be at most a constant number of facets contained completely inside C. Therefore, at least $\Omega(k)$ facets intersect both e and C. A facet can intersect C in two ways - inward and outward, depending on whether the edges intersection C tend to converge or diverge after crossing C (Figure 3.12). Consider a facet f (containing triangle t') that intersects C only in the inward direction. Either more than half of t' is contained in C or the arc length of some intersection between C and f is $\Omega(1)$. Only a constant number of facets can have only inward intersection. Therefore, $\Omega(k)$ facets intersect C outwards, and (by a Markov argument) $\Omega(k)$ of these intersections have arc length $< k^{-1}$. Consider any such facet f' - the two

edges intersecting C has a minimum distance of less than k^{-1}. Since f' intersects e, the angle between the edges must be $O(k^{-1})$, and f' is $O(k^{-1})$-thin. This shows that for any triangle t generating k faces, $\Omega(k)$ of these faces are $O(k^{-1})$-thin facets. We will say that t is a *witness* for these $O(k^{-1})$-thin facets, since the intersection of C with these facets shows their $O(k^{-1})$-thinness. For any facet f of min-thinness α, note that at most $(\alpha n^{1-v'})^{-1}$ triangles are witnesses for *any* thinness (since the minimum distance between sharp edges is $< n^{-(1-v')}$ and the angle is α.)

We sum up the constributions (in terms of complexity) of all triangles generating more that $n^{1-v'}$ faces. Let this sum be S. By the arguments given above, βS comes from thin facets that are witnessed (for some fixed constant $\beta < 1$). Some facets are counted more than once in βS because many triangles can witness one facet. Let us consider all witnessed facets with min-thinness in the range $[\alpha/2, \alpha]$. There are at most $O(\alpha n)$ such facets and each is counted in S at most $2(\alpha n^{1-v'})^{-1}$ times. Therefore, the total contribution of this in S is $O(n^{v'})$. We apply this argument for the values $\alpha = n^{-(1-v')}, 2^{-1}n^{-(1-v')}, 2^{-2}n^{-(1-v')}, \cdots, (cn)^{-1}$ (for some sufficiently large constant c) and get that $S = O(n^{v'} \log n)$.

□

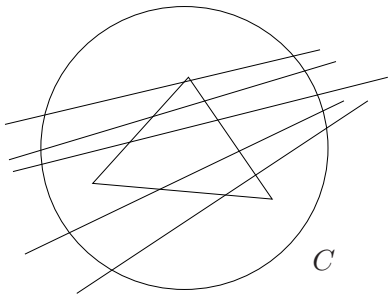

Figure 3.11: Facets intersecting with C

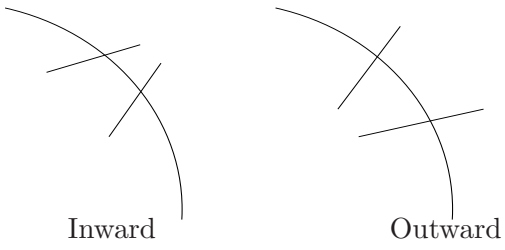

Figure 3.12: Inward and Outward

Noting that $u = 1 - a/2$ and ensuring $v > 1 - a/4$, we can set $v' = (1 + u)/2$. By the previous claim, the complexity of the lifted fence can be made $\widetilde{O}(n^{1-a/4})$, proving Lemma 3.2.8.

3D-Terrain-Filter

if $b(\mathcal{D}) = 1$
  then convexify $\mathcal{D}$ using
    Offline-Reconstruction
  else
    (1) build fence $\mathcal{F}$ and convexify
    into $\mathcal{F}^c$
    (2) identify patch containing query face f
      convexify extended patch
      with Offine-Reconstruction

3.2.5 Online reconstruction

We now have all the necessary tools required to make the filter. By Theorem 3.2.2, we estimate the distance to convexity in $\widetilde{O}(n^{1+\alpha-\beta})$ time. If $b(\mathcal{D}) = 1$, then we convexify the terrain in $\widetilde{O}(n)$ time by appealing to Theorem 3.2.1. Since $\varepsilon_{\mathcal{D}} < n^{-\alpha}$, the running time is $\widetilde{O}(\varepsilon_{\mathcal{D}}^{-\alpha^{-1}})$.

Assume now that $b(\mathcal{D}) = 0$, which implies that $\varepsilon_{\mathcal{D}} > n^{-\beta}$. By Lemma 3.2.9, setting $r = n^{\beta}$ shows that $\kappa(\mathcal{F}^c, X) \leq O(\varepsilon_{\mathcal{D}} n)$. A crucial aspect of the reconstructed fence is that the convexification of any patch can be done in isolation, as long as we include the fence triangles bounding the patch in question. This follows from this transitivity lemma, which we prove later. (We use the subscript xy to denote the projection onto the xy-plane.)

Lemma 3.2.13. *Let f, g be two faces of a (possibly discontinuous) terrain and let F, G be two sets of faces in convex position such that: (i) removing the region F_{xy} disconnects f_{xy} from g_{xy}; same is true of G_{xy}. If f (resp. g) is in convex position with F (resp. G), then f and g are in convex position with each other.*

Given a query f, unless $b(\mathcal{D}) = 1$, the filter finds the patch corresponding to f. The random seed is used only to compute $b(\mathcal{D})$ and build the convexified fence $\mathcal{F}^c$. n other words, all queries use the same convexified fence $\mathcal{F}^c$, which ensures consistency between all the outputs. The filter then proceeds to reconstruct the entire patch together with all its bordering fence triangles (what we call the *extended patch*). By Lemma 3.2.6, computing the fence takes $\widetilde{O}(n^{1-a/2})$ time. By our setting of $r = n^{\beta}$, to find Σ requires $\widetilde{O}(r^2 n^{1-a/4}) = \widetilde{O}(n^{1+2\beta-a/4})$ time, and convexification can be done in time $\widetilde{O}(n^{1+2\beta-a/4})$. Reconstructing the fence adds nothing to the asymptotic complexity. Convexifying the corresponding patch takes $\widetilde{O}(n^a)$. Putting everything together, we see that the time for answering a query is

$$\widetilde{O}(n^{1+\alpha-\beta} + \varepsilon_{\mathcal{D}}^{-\alpha^{-1}} + n^{1-a/2} + n^{1+2\beta-a/4} + n^a)$$

The constraints $0 < a < 1$, $0 < \alpha < \beta < 1$, and $\alpha \geq \frac{1}{2}(3\beta - 1)$ are all satisfied if we set α to be an arbitrarily small positive constant and $a = 12/13$ and $\beta = 1/13$.

Theorem 3.2.14. *Any n-face 3D bounded aspect ratio terrain $\mathcal{D}$ has a convexity filter with a worst case query time of $O(n^{12/13+\alpha} + \varepsilon_{\mathcal{D}}^{-O(\alpha^{-1})})$.*

We now prove Lemma 3.2.13. We will prove the following claim, from which Lemma 3.2.13 will be obvious.

Claim 3.2.15. *Let f, g be two faces of a possibly discontinuous terrain and S be a set of faces in convex position. If removing S_{xy} disconnects f from g and f_{xy} and g_{xy} are in convex position with S, then f and g are in convex position with each other.*

Proof. Let $\mathcal{B}$ be the (possibly unbounded) convex body formed by the faces of S. For simplicity, assume that S is minimal - we cannot remove any face from S and still maintain the lemma assumptions. Take the region in the xy-plane disconnected by the removal of S_{xy}. Project the boundary of this region onto $\mathcal{B}$ and call this curve C^S - note that the curve C^S_{xy} separates f_{xy} from g_{xy}. By the minimality assumption, the curve C^S is simple. Wlog, let f_{xy} be contained in the inner region defined by C^S_{xy}. Let H^f be the plane containing f and C^f be the intersection curve of H^f and $\mathcal{B}$. Note that C^S lies completely to one side of H^f. Suppose there is a vertex of g that lies in the halfspace (defined by H^f) that does not contain C^S. Note that g lie inside $\mathcal{B}$, since it is in convex position with S. Therefore, it must be the case that the xy-projection of this vertex lies inside C^f, which lies inside C^S. This is contradicts the fact that C^S_{xy} separates f_{xy} from g_{xy}. The face g lies completely to one side of H^f.

Defining H^g and C^g (which may not be closed), we can apply a similar argument to show that f lies completely on one side of H^g.

□

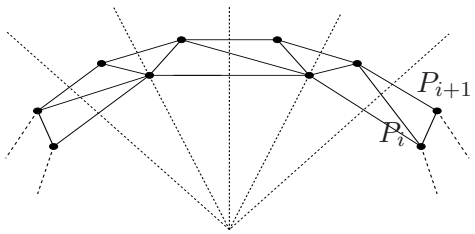

Figure 3.13: The concentric rings of the xy-projection.

3.2.6 Lower bound

We show that any 3D convexity filter has a worst case query time of $\Omega(\varepsilon_{\mathcal{D}}^{-1})$ time, thus revealing a fundamental complexity gap between the two and three-dimensional cases. Recall that the 2D filter made essential use of a certain transitivity feature of convexity violation: if e, f, g are edges in clockwise order and (e, g) is not in convex position, then at least one of (e, f) or (f, g) is not either. In designing our filter, we used a 3D variant of this by letting the fence play the role of f. But, unlike in 2D, the fence cannot be a constant size object. Why this implies a lower bound is explained below.

We appeal to Yao's *minimax lemma* to deal with the fact that our algorithms are randomized. We will start with $\varepsilon_{\mathcal{D}} = \Theta(\log n/n)$. After describing the construction for this value, we will show how to handle any value of $\varepsilon_{\mathcal{D}} = \Omega(\log n/n)$. Assume that the filter changes at most $c\varepsilon_{\mathcal{D}} n$ faces, for some fixed $c > 1$. We define a distribution of inputs and show that, for any deterministic algorithm that performs reconstruction, some query takes $\Omega(n/\log n)$ expected time over that distribution.

We start with a fixed $\mathcal{D}$ and build the distribution around it. Fix some parameter $m > 0$. The xy-projection of $\mathcal{D}$ consists of $\Theta(\log m)$ concentric regular polygons $P_0, P_1, \cdots, P_{k-1}$ centered at the origin (Figure 3.13): (i) the innermost polygon P_0 has a constant number of vertices; (ii) P_i has $2^{i-1}|P_1|$ vertices and every other edge is parallel to an edge of P_{i-1}; (iii) P_{k-1} has $m/(c_1 \log m)$ vertices, for fixed $c_1 > 0$. The radii of the P_i's are chosen so that the boundaries are fairly close to each other but disjoint. Next, we lift these polygons vertically so that their edges are all horizontal tangents to the paraboloid $\mathcal{C} : Z = -(X^2 + Y^2)$ at their midpoints (Figure 3.14). Each band between consecutive polygons is triangulated appropriately and the construction is lifted to $\mathcal{C}$ to form a convex terrain with (lifted) P_0 as its highest face. Finally, we add an extra polygon P_k that is a slightly scaled-up version of P_{k-1}. The band between P_{k-1} and P_k consists of $m/(c_1 \log m)$ trapezoids, each one of which is now divided up into a stack of $c_1 \log m$ parallel subtrapezoids. After lifting, each subtrapezoid finds itself tangent to $\mathcal{C}$. Setting $m = \Theta(n)$, triangulating all faces produces n faces.

The terrain $\mathcal{D}$ is convex: we introduce convexity violations by choosing one *stack* $\mathcal{S}$ of subtrapezoids, and tilting them ever so slightly so that: (i) each subtrapezoid violates one common triangle of P_1; (ii) the stack $\mathcal{S}$ violates $O(1)$ triangles per (P_i, P_{i+1}) band. This tilting is done so that the common edge between this stack of subtrapezoids and the trapezoid of P_{k-1} does not move (the tilting is done with this edge hinged). Once this tilting is done, there will have to be slight modifications performed on this stack and the stacks adjacent to $\mathcal{S}$ to ensure that all stacks are in convex position with each other. Note that $\varepsilon_D n < c' \log n$ (for some constant c'). Suppose we decide to keep all the stacks of subtrapezoids. There are only $O(1)$ triangles in each (P_i, P_{i+1}) band which violate convexity with $\mathcal{S}$. Since there are $O(\log n)$ bands, the total number of faces which are not in convex position with the stack are $c' \log n$. All the remaining faces are in convex position with each other. This constant c' is independent of c_1 - in other words, we can make c_1 arbitrarily larger than c'.

The filter guarantees to change at most $cc' \log n$ faces. Set $c_1 \log m > cc' \log n > c\varepsilon_{\mathcal{D}} n$. In this way, a query to the common violating triangle of P_1 cannot return the triangle unchanged. Indeed, if it did, then the entire stack $\mathcal{S}$ of $c_1 \log m$ triangles would later have to be modified, which would prove the filter faulty. Modifying the violating triangle of P_1 appropriately requires knowing where the stack $\mathcal{S}$ is placed around the (P_{k-1}, P_k) band, which takes $\Omega(|P_k|)$ expected time.

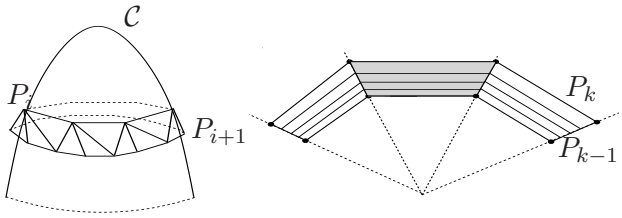

Figure 3.14: A hard terrain to reconstruct.

The extension to higher values of ε is quite straightforward. Essentially, the number of concentric rings is (upto constant factors) equal to εn. Fix some parameter m. We choose concentric regular polygons $P_0, P_1, \cdots, P_{k-1}$ (where $k = \Theta(\varepsilon m)$) such that P_0 has a constant number of vertices, P_i has either $|P_{i-1}|$ or $2|P_{i-1}|$ vertices, and P_{k-1} has $(c_1\varepsilon_{\mathcal{D}})^{-1}$ vertices (for some sufficently large constant c_1). The outermost polygon P_k is as before a slightly scaled up version of P_{k-1} and the band (P_{k-1}, P_k) consists of stacks of subtrapezoids (as before) with stack size of $c_1\varepsilon_{\mathcal{D}} m$. One of these stacks is tilted to ensure that it violates one common triangle of P_1 but violates the convexity of at most $O(1)$ subtrapezoids in each ring. The parameter $m = \Theta(n)$ is chosen to ensure that the total number of faces is n. The distance to convexity $\varepsilon_{\mathcal{D}}$ is $\Theta(\varepsilon)$. Using the same argument as above, we can force a query for a common violating triangle in P_1 to make a modification. For this, the tilted stack of subtrapezoids must be detected, which will take $\Omega(P_k) = \Omega({\varepsilon_{\mathcal{D}}}^{-1})$ time.

Theorem 3.2.16. *Any convexity filter for a terrain $\mathcal{D}$ of n faces has a worst case query time of $\Omega(\varepsilon_{\mathcal{D}}^{-1})$ for any n such that $(\log n)/n \leq \varepsilon_{\mathcal{D}}$.*

Chapter 4

Expansion Reconstruction

In our final result on reconstruction, we discuss the problem of expander reconstruction. Given an input bounded degree graph G, the aim is to output a graph on the vertices of G that has high conductance and is as close to G as possible. The queries are made for vertices and the filter outputs, in sublinear time, the neighbors of this vertex (in the output graph).

Property testers for expansion in bounded degree graphs were first discussed in [49]. One of the observations made was that the mixing rate of random walks in graphs could be estimate accurately in sublinear time. We take these techniques further and show that there are many other interesting (random walk related) properties that can be estimated.

This work deals with an independent concept that has no connections with sublinear algorithms. It is well known [58, 64] that random walks mix rapidly in expanders. Suppose we add some "noise" in the form of a small arbitrary graph connected arbitrarily to an expander. What can we say about the random walks here? We would expect that the mixing properties of the expander have not been affected too much. We make this formal and prove that this is indeed the case. A lot of combinatorial and probabilisitic tools have been used to achieve this.

4.1 Preliminaries

The input is a large graph $G = (V, E)$, with n vertices. We assume that all the vertices have degrees bounded by some specified constant d, which we assume to be sufficiently large (say at least 10). The graph G is represented by *adjacency lists*: for every vertex v, there is list of vertices (of size at most d) adjacent to v. Given a subset of vertices $S \subseteq V$ of size at most $n/2$, let $\bar{S} = V \setminus S$, and let $E(S, \bar{S})$ denote the set of edges crossing the cut $(S, \bar{S})$. The expansion of the cut is defined to be $|E(S, \bar{S})|/|S|$. The expansion of the graph is the minimum expansion over all cuts in the graph. An expander is a generic term for a graph with high expansion.

We consider a related parameter, the conductance of a cut, which is defined to be just the expansion of the cut divided by $2d$. The conductance of the graph is its expansion divided by $2d$. We are given a parameter $0 < \phi < 1$, and the input graph

is supposed to (but may not) have the property that its conductance is at least ϕ. We think of ϕ as a small constant.

The filter is represented by a deterministic sublinear time procedure RECONSTRUCT takes a vertex v and the random seed s as inputs and outputs at most d neighbors of v in G' (the adjacency list of v). To allow for parallel processing of queries, we assume that a sublinear size random seed s is fixed at the beginning and used for all queries.

All the answers of RECONSTRUCT are consistent (thereby giving a description of the graph G'). The final graph G' has conductance at least ϕ' for some $\phi' < \phi$, degrees bounded by d, and the number of edges changed (from G to get G') is at most a small factor more than the optimal number of edges that need to be changed to make the conductance at least ϕ. All these properties are satisfied with high probability over the choice of the random seed s, over all possible queries v.

We now state our main theorem regarding the properties of our filter:

Theorem 1. *There is a determinisitic procedure* RECONSTRUCT *that takes as input a vertex v and random seed s of length*[1] $\tilde{O}(\sqrt{n}/\phi^2)$*, and outputs the adjacency list of v in G'. The following properties hold with probability at least $1 - 1/n^2$:*

1. *Queries can be answered in parallel, and each answer takes $\tilde{O}(\sqrt{n}/\phi^2)$ time,*
2. *All query answers are consistent: the vertex u is output as a neighbor of v iff v is output as a neighbor of u,*
3. *The final graph G' has conductance at least $\phi' = \Omega(\phi^2/\log n)$ and degree bound d,*
4. *The number of edges changed is at most $O(\frac{1}{\phi} OPT)$, where OPT is the optimal number of edges needed to be changed to make the conductance at least ϕ.*

Note that the parallel filter can be used as a property tester for expansion, so by the lower bound for testing expansion by Goldreich and Ron [50], the running time of our filter is optimal upto $\operatorname{poly}\log(n)$ factors.

4.1.1 Comparison to property testing : why is reconstruction hard?

The input model of adjacency lists for sparse graphs was introduced by Goldreich and Ron [50], where they designed testers for a large set of problems. More general results about the testability of large classes of properties were obtained by Czumaj and Sohler [34] and by Benjamini, Schramm, and Shapira [21]. The technique of using random walks for testing was first introduced by Goldreich and Ron [48] for testing bipartiteness.

Goldreich and Ron [49] formally posed the problem of expansion testing (by a result in [50], there is a lower bound of $\Omega(\sqrt{n})$ queries to the input graph). Given a d-degree bounded input graph G and a distance parameter ε, we want to distinguish

[1] We use the $\tilde{O}$ notation to suppress dependence on $\operatorname{poly}\log(n)$ factors.

between the case that G is an expander, and that at least εnd edges need to be changed in G to make it an expander[2]. The first progress towards this was made by Czumaj and Sohler [35]. Further work on testing expansion improving certain parameters was done in [55, 67].

The reconstruction problem is a much harder problem than property testing for the obvious reasons: first, the property testing problem is a decision problem, whereas reconstruction algorithm needs to actively take local action in each query to fix the input graph, and patch up sparse cuts. Second, in the notation of Theorem 1, property testing algorithms only need to distinguish between graphs which have $\text{OPT} = 0$ and $\text{OPT} \geq \varepsilon nd$, whereas reconstruction algorithms are required to approximately compute OPT, for all values of OPT.

More importantly, the known techniques in property testing algorithms are inadequate for reconstruction. Such approaches show that in a graph G that is far from being an expander, there exists many "bad" vertices starting from which random walks in G do not mix rapidly. One may expect that adding d edges at random to all such bad vertices will suffice to make it an expander, and indeed it does, but it may be an overkill: there are graphs G where all vertices as bad, yet G can be made an expander by adding very few edges. Think of the case where G consists of two disjoint expanders - one of size $(1-\delta)n$ and the other of size δn (where δ is a very small constant). We only need to add random edges to every vertex in the *smaller* expander to patch the bad cut. But all present testing algorithm would deem *all* vertices as "bad".

We therefore need to devise completely new techniques for reconstruction. The main approach to reconstruction is to identify *weak* and *strong* vertices. Intuitively, weak vertices are those that lie on the smaller side of bad cuts, and therefore need edges to be added from them, whereas strong vertices are those that are part of a large expanding component (if one exists). The reconstruction procedure should be careful to only add edges to the weak vertices.

The property testing approach is to distinguish between weak and strong vertices by thresholding the distance from mixing for a short random walk. All the property testing results are obtained using the $\mathcal{L}_2$-distance, but this is too sensitive to noise, harming the mixing properties of many strong vertices. Thus, we develop completely new techniques based on the $\mathcal{L}_1$ distance, and prove noise-tolerance properties of Markov chains under this norm. These properties are crucial in our reconstruction algorithm.

4.2 Noise-tolerance of Markov Chains

First, we discuss our theorem about the noise-tolerance of Markov chains. Later, we apply this to our setting of expansion reconstruction. Let M be a finite Markov chain with state set V and transition probabilities p_{uv} for $u, v \in V$. The k-step transition probabilities will be denoted as p^k_{uv}. The vector p^k_u represents the probability

[2]This is not the complete formulation, but it suffices for this discussion.

distribution on V for a k-step random walk starting from u (for $k = 0$, $p^0_{uu} = 1$ and $p^0_{uv} = 0$ for $v \neq u$). For simplicity, we assume that M is irreducible, and that for any $u \in V$, $p_{uu} \geq 1/2$ (so that the chain is aperiodic). In this case, there is a unique stationary distribution π for the Markov chain. For a subset of states $S \subseteq V$, define $\pi(S) = \sum_{u \in S} \pi_u$. Note that the chain need not be reversible.

The conductance of the Markov chain is defined to be the largest number ϕ such that for any subset of states $S \subseteq V$,

$$\sum_{u \in S, v \in V \setminus S} \pi_u p_{uv} \;\geq\; \phi \cdot \min\{\pi(S), \pi(V \setminus S)\}.$$

It is a well-known fact (see, for example [58, 64, 75]) that if the Markov chain has high conductance, then from any starting state, the chain converges to the stationary distribution exponentially fast at a rate determined by the conductance.

Suppose we are given a Markov chain which *almost* has high conductance, in the following sense. There is a "large" subset of states $V' \subseteq V$ such that the chain restricted to the subset V' has high conductance, and nothing can be said about the remaining "small" part of the state space, $B = V \setminus V'$ (the "noise"). Then, we would like to show that except for a set of states B' of stationary measure comparable to that B, from all other starting states the chain will have some fast mixing properties (i.e. the noise has limited influence).

This turns out to be hard to prove because it can be true for completely different reasons. Suppose that the noise B is almost disconnected from V'. Then, a random walk starting from V' will almost never leave V' and since V' has high conductance, will definitely mix rapidly inside V'. On other other hand, suppose that the whole chain has high conductance (not just restricted to V'). In this case, although walks from inside V' will encounter B, all walks will still mix rapidly. We need a proof that can interpolate between these scenarios. A first approach to proving this would be to estimate the probability that a walk from V' never hits B. But such a bound would be too weak: there are many states (compared to $\pi(B)$) which are sufficiently "close" to B that a random walk from them will almost certainly hit B.

We now proceed to formalize the setting. Given a subset of states $V' \subseteq V$, define the V'-conductance of the chain to be the largest number ϕ such that for any set $S \subseteq V'$,

$$\sum_{u \in S, v \in V' \setminus S} \pi_u p_{uv} \;\geq\; \phi \cdot \min\{\pi(S), \pi(V' \setminus S)\}.$$

We also need the notion of a *uniform averaging walk* in the Markov chain of ℓ steps: in such a walk, we choose a number $k \in \{0, 1, 2, \ldots, \ell - 1\}$ uniformly at random, and stop the chain after k steps. Thus, the distribution of the final state of the uniform averaging walk starting from u is $\bar{p}^\ell_{uv} = \sum_{k=0}^{\ell-1} p^k_{uv}$. The *total variation distance* between two distributions ξ and ψ on the states is defined to be $\|\xi - \psi\|_{\mathrm{TV}} = \max_S |\xi(S) - \psi(S)| = \frac{1}{2}\|\xi - \psi\|_1$, where S is an arbitrary subset of states, and $\xi(S)$ and $\psi(S)$ are, respectively, the measures of S under ξ and ψ respectively.

Theorem 2. *Let the Markov chain M have V'-conductance ϕ. Let $\pi_0 = \min\{\pi_u/\pi(V') : u \in V'\}$. Then, there is a set B' such that $\pi(B') \leq 2\pi(B)$ with the property that starting from any state $s \in V \setminus B'$, if the uniform averaging Markov chain of $\ell \geq 8\log(1/\varepsilon\pi_0)/(\varepsilon\phi^2)$ steps is run, then the final probability distribution $\bar{p}_s^\ell$ satisfies $\|\bar{p}_s^\ell - \pi\|_{TV} \leq \varepsilon + \pi(B)$.*

We prove this theorem as follows. We consider a closely related Markov chain that is restricted to V'. We retain all the original transitions in M between any pair of vertices $u, v \in V'$ with the same probabilities. We assign all such transitions a cost of 1. The meaning of this cost will be explained later.

For any pair of vertices $u, v \in V'$, and for every integer $j \geq 2$, define q_{uv}^j to be the total probability of all length j walks from u to v all of whose states, except for the end points u and v, are in B. Then we add a new transition e_{uv}^j from u to v with cost j and probability q_{uv}^j. Since M is irreducible, any walk in M that enters B has to eventually leave it, and hence the new transitions define a Markov chain on V'. Call this new Markov chain M'. Since M is irreducible, so is M'. The chain M' is called an *induced* Markov chain by Aldous and Fill [8].

Now, for any walk in M', define the cost of the walk to be the total cost of all transitions in the walk. The cost of a walk in M' is naturally mapped to the length of a corresponding walk taken in M (where taking one of the new transitions is to be interpreted as taking *all* the corresponding walks of length equal to the prescribed cost which are entirely in B, except for the end points).

This correspondence between walks in M and M' also implies that the stationary distribution in M' is one that assigns probability $\pi'_u = \pi_u/\pi(V')$ to state u. This can be seen by considering an arbitrary state $z \in V'$ and using the fact that the stationary probability of any state $u \in V'$ is proportional to the expected number of visits to u before returning to z, and then using the correspondence between the walks to argue that the expectation is the same in both M and M'. For convenience, for $u \in B$, we define $\pi'_u = 0$.

Lemma 1. *For any integer $t > 0$, there exists a set $\tilde{B} \subseteq V'$ such that $\pi(\tilde{B}) \leq \pi(B)$, with the property that for all $v \in V' \setminus \tilde{B}$,*

$$\mathbb{E}[\text{cost of } t \text{ step walk in } M' \text{ from } v] \;\leq\; 2t.$$

Proof. Fix any starting state $s \in V'$. Let X^k be the k-th state on a t step walk in M' from s. For any $u \in V'$, let p'^k_{su} be the probability of reaching u from s on step k of a random walk in M'.

For any $u \in V'$, we have $\mathbb{E}[\text{cost of step } k{+}1 \mid X^k = u] = \mathbb{E}[\text{cost of one step from } u]$, by the Markov property. So define

$$c_u \;:=\; \mathbb{E}[\text{cost of one step from } u] = \sum_{v \in V'} \left(p_{uv} + \textstyle\sum_{j \geq 2} q_{uv}^j \cdot j \right)$$

We have

$$\begin{aligned}\mathbb{E}[\text{cost of } t \text{ step walk from } s] \;&\leq\; t + \sum_{k=1}^{t} \sum_{u \in V'} \Big[\mathbb{E}[\text{cost of step } k+1 \mid X^k = u] - 1\Big] \cdot {p'}^k_{su} \\ &=\; t + \sum_{k=1}^{t} \sum_{u \in V'} (c_u - 1) \cdot {p'}^k_{su}.\end{aligned}$$

Now, we define

$$\tilde{B} \;:=\; \left\{ s \in V' : \sum_{k=1}^{t} \sum_{u \in V'} (c_u - 1) \cdot {p'}^k_{su} \;\geq\; t \right\}. \tag{4.1}$$

With this definition, it is clear that for any starting state $s \in V' \setminus \tilde{B}$, the expected cost of an t step walk from s is at most $2t$, as required by the statement of the lemma. We proceed to bound $\pi(\tilde{B})$ as follows:

$$\begin{aligned} t \cdot \pi(\tilde{B}) \;&\leq\; \sum_{s \in V'} \pi_s \cdot \left[\sum_{k=1}^{t} \sum_{u \in V'} (c_u - 1) \cdot {p'}^k_{su} \right] \\ &=\; \sum_{k=1}^{t} \sum_{u \in V'} (c_u - 1) \cdot \sum_{s \in V'} \pi_s {p'}^k_{su} \\ &=\; \sum_{k=1}^{t} \sum_{u \in V'} (c_u - 1) \cdot \pi_u \\ &=\; t \cdot \sum_{u \in V'} \pi_u (c_u - 1). \end{aligned}$$

where the equality on the second line follows because $\pi' = \pi' {P'}^k$, where P' is the transition matrix of M', which implies that $\sum_{s \in V'} \pi'_s {p'}^k_{su} = \pi'_u$, and because $\pi'_u \propto \pi_u$ for $u \in V'$.

We now need to bound $\sum_{u \in V'} \pi_u (c_u - 1)$. Note that c_u is exactly the expected time to return to the set V' starting from u in the original Markov chain M. Since M is irreducible, by Kac's lemma (see [8]), we have

$$\begin{aligned} \sum_{u \in V'} \pi_u c_u \;&=\; 1 \\ \implies \sum_{u \in V'} \pi_u (c_u - 1) \;&=\; 1 - \pi(V') = \pi(B) \end{aligned}$$

Thus, we have $t \cdot \pi(\tilde{B}) \leq t \cdot \pi(B)$, which implies that $\pi(\tilde{B}) \leq \pi(B)$. □

Now, we would like to prove that the set $B' = \tilde{B} \cup B$, where $\tilde{B}$ is defined in Lemma 1 works for Theorem 2, when t is chosen to be the mixing time of M', which can be bound in terms of the conductance of M'. The idea is that even without

the newly added transitions, the Markov chain V' has high conductance. Thus, if a short walk in V' mixes, then by Lemma 1, a corresponding short walk in the original chain should mix as well.

Proof. (Theorem 2) We use the notion of *stopping rules* for Markov chains [59]. A stopping rule is a rule that observes the walk and tells us whether to stop or not, depending on the walk so far (but independently of the continuation of the walk). This decision may be reached using coin flips, so the stopping rule has to only specify for each finite walk w, the probability of continuing the walk, so that with probability 1, the walk is stopped in a finite number of steps. The expected time for a stopping rule to terminate the walk yields bounds on the mixing time of the chain.

Let $B' = \tilde{B} \cup B$, where the set $\tilde{B}$ is as defined in equation (4.1) in Lemma 1. Let $s \in V \setminus B' = V' \setminus \tilde{B}$. We consider a random walk in the original Markov chain M starting from s with the following probabilistic stopping rule Γ: stop the walk as soon as it has taken t steps in the induced walk in M'. Note that this stopping rule always stops the walk on some state in V'. Denote by $\mathbb{E}_s[\Gamma]$ the expected number of steps the walk takes starting from s before being terminated by Γ.

Now, because the Markov chain M has V'-conductance at least ϕ, the Markov chain M' has conductance at least ϕ, and hence by the results of [58, 64], a $t = 2[\log(1/\varepsilon\pi_0)]/\phi^2$ step walk starting from s mixes on V', i.e. $\|{p'}_s^t - \pi'\|_{\mathrm{TV}} \leq \varepsilon/2$, where ${p'}_s^t$ is the probability vector for a t step random walk starting at s. Furthermore, we have $\|\pi' - \pi\|_{\mathrm{TV}} = \pi(B)$, and hence by the triangle inequality, $\|{p'}_s^t - \pi\|_{\mathrm{TV}} \leq \varepsilon/2 + \pi(B)$. Now, by Lemma 1, we have $\mathbb{E}_s[\Gamma] \leq 2t$. Thus, for all $s \in V \setminus B'$, the stopping rule Γ stops the walk in a distribution that is within total variation distance $\varepsilon/2 + \pi(B)$ of the stationary distribution within an expected $2t$ steps.

Lovász and Winkler [59] define $\mathcal{H}(s, d_\delta)$ to be the minimal expected time for a stopping rule to stop a chain started from s in a distribution that is within variation distance δ of the stationary distribution, and thus we have shown that $\mathcal{H}(s, d_{\varepsilon/2+\pi(B)}) \leq 2t$. Now, Theorem 4.22 in [59] implies that for the uniform averaging chain of ℓ steps in M started from s, if $\bar{p}_s^\ell$ is the final distribution, then

$$\|\bar{p}_s^\ell - \pi\|_{\mathrm{TV}} \;\leq\; \varepsilon/2 + \pi(B) + \frac{1}{\ell}\mathcal{H}(s, d_{\varepsilon/2+\pi(B)}) \;\leq\; \varepsilon + \pi(B),$$

since $\ell \geq 4t/\varepsilon$, which completes the proof. □

4.3 The Reconstruction Algorithm

We give an overview of the reconstruction algorithm. In the first step, the algorithm tries to identify vertices that are part of a low conductance cut (a "bad" cut), and in the second step, it attempts to boost the conductance of the cut by adding edges to the identified vertices. We use a random walk based procedure to separate vertices on the smaller side of a bad cut from the larger. The details of this separation procedure are given in Section 4.3.1.

One way to fix the bad cuts is to add random edges to the identified vertices. This works, but doesn't allow for parallel query processing which requires the randomness to be fixed in advance. Instead, we use an explicit expander, and construct a hybrid of the original graph and the expander. The hybridization is locally done to allows parallel query processing. Details of the hybridization procedure are given in Section 4.3.2.

4.3.1 Separating vertices

We define a Markov chain on the graph, and study random walks in this Markov chain. The states of the chain are the vertices of the graph, and the edges represent transitions, such that from each vertex u, any outgoing edge (u, v) is taken with probability $p_{uv} = 1/2d$. With the remaining probability, the walk stays at the current vertex. Note that this is an irreducible (assuming the graph is connected), aperiodic, and time-reversible chain, and the stationary distribution is uniform on all the vertices.

Let $\ell = c\log(n)/\phi^2$, where c is a sufficiently large enough constant. We will consider the *uniform averaging walk* of length ℓ, in two forms based on related probabilistic stopping rules:

1. Pick an integer $t \in \{0, 1, 2, \ldots, \ell - 1\}$ uniformly at random, and stop the walk after t steps. This is the uniform averaging walk of length ℓ. Let q_{uv} be the probability of reaching v from u after such a random walk (i.e. $q_{uv} = \bar{p}^{\ell}_{uv}$, in the notation of Section 4.2).

2. Pick two integers $t_1, t_2 \in \{0, 1, 2, \ldots, \ell - 1\}$ uniformly at random, and stop the walk after $t_1 + t_2$ steps. Let Q_{uv} be the probability of reaching v from u after such a random walk.

Let q_u be the probability vector of q_{uv}'s. For a subset of vertices S, let $q_u(S) := \sum_{v \in S} q_{uv}$ (similarly we define Q_u and $Q_u(S)$). It is easy to see that for any two vertices u and v, we have $Q_{uv} = \sum_w q_{uw} q_{wv} = \sum_w q_{uw} q_{vw}$, because $q_{wv} = q_{vw}$, as all edges have the same probability of $1/2d$. For ease of notation, we will refer to the above walks as q-random walks and Q-random walks. In our procedures and definitions, we will use some constants: c is a sufficiently large integer and $\alpha, \beta, \gamma, \delta > 0$ are sufficiently small (values $\alpha = 1/1000, \beta = 1/10, \gamma, \delta = 1/100$ work [3]).

We now define *weak* and *strong* vertices. Intuitively, strong vertices are those that can reach a vast majority of vertices with probability $\Omega(1/n)$ through a short random walk. Formally, we will look at the mixing properties of random walks in the $\mathcal{L}_1$ norm. Note that there can be vertices that are neither strong nor weak. In the following definition, $\vec{1}$ is the all 1's vector.

Definition 1 (Strong and Weak vertices)**.**

[3] We have not optimized these constants.

1. *A vertex $u \in V$ is called* strong *if* $\|q_u - \vec{1}/n\|_{TV} \leq \alpha$ $(= 1/1000)$.

2. *A vertex $u \in V$ is called* weak *if* $\|Q_u - \vec{1}/n\|_{TV} \geq 1/4$.

Intuitively, strong vertices are those that reach a vast majority of vertices (through short q-walks) with probability $\Omega(1/n)$. The basic idea is to perform many q-walks from u (and v) and compute the number of collisions between these walks (at the endpoints): this is a twist on the idea of Goldreich and Ron [49] for using random walks to estimate the mixing of a random walk. A birthday paradox like argument suggests that $O(\sqrt{n})$ walks should be sufficient to estimate probabilities of $\Omega(1/n)$. Unfortunately, assuming nothing about the vectors q_u and q_v, we cannot get a reasonable bound on the variance of our randomized estimate[4]. For this reason, we define the *reduced collision probability*, which disregards collisions that occur due to vertices reached with extraordinarily high probability. This quantity attempts to approximate Q_{uv} and can be estimated in sublinear time. For the purpose of separating weak vertices from strong ones, it suffices to estimate these probabilities.

We give a detailed exposition of the procedures that are used to separate weak vertices from strong ones. The main procedure is based on approximating the reduced collision probability values. For technical reasons (which should become clear soon), we will introduce some extra parameters to the definition of the reduced collision probability. This will help us deal with the errors introduced by the restriction of sublinear time.

Definition 2 (Reduced Collision Probability)**.** *Consider vertices u, v. Let $S_u^\sigma = \{w : q_{uw} \leq (1-\sigma)/\sqrt{n}\}$ (set S_v^σ is similarly defined). The* σ-reduced collision probability *of vertices u, v (denoted by r_{uv}^σ) is:*

$$r_{uv}^\sigma = \sum_{w \in S_u^\sigma \cap S_v^\sigma} q_{uw} q_{vw}$$

We will set σ to be some small constant (say 1/4). We prove some useful properties of strong vertices.

Lemma 2. *The following properties of strong vertices hold:*

1. *For any strong vertex v, there can be at most $\sqrt{\alpha}n$ vertices with $q_{uv} \leq (1 - \sqrt{\alpha})/n$.*

2. *For any strong vertex u and any $\sigma \leq \frac{1}{2}$, we have $q_u(S_u^\sigma) \geq 1/2$.*

3. *Let u and v be strong vertices. Then for any $\sigma \leq \frac{1}{2}$, we have $r_{uv}^\sigma \geq (1-\beta)/n$ for $\beta \geq 3\sqrt{\alpha}$.*

[4]Think of the extreme situation where for some w, $q_{uw} = 1 - 1/n - 1/n^2$, and $q_{uz} = 1/n^2$ for $z \neq w$; whereas all $q_{vz} = 1/n$. Although $Q_{uv} = \Omega(1/n)$, it is very unlikely that a sublinear time procedure can detect a collision between q-random walks from u and v.

Proof. The first part follows immediately from the definition of strong vertices. For the second part, note that there can be at most $2\sqrt{n}$ vertices in $V \setminus S_u^\sigma$. Thus, there are at least $(1-\sqrt{\alpha})n - 2\sqrt{n}$ vertices $v \in S_u^\sigma$ such that $q_{uv} \geq (1-\sqrt{\alpha})/n$, and hence

$$q_u(S_u^\sigma) \;\geq\; [(1-\sqrt{\alpha})n - 2\sqrt{n}] \cdot (1-\sqrt{\alpha})/n \;\geq\; 1/2.$$

We now prove the third part. There are at least $(1-\sqrt{\alpha})n - 2\sqrt{n}$ vertices $w \in S_u^\sigma$ such that $q_{uw} \geq (1-\sqrt{\alpha})/n$, and a similar statement holds for v. Thus, there are at least $(1-2\sqrt{\alpha})n - 4\sqrt{n}$ vertices in $S_u^\sigma \cap S_v^\sigma$ such that both q_{uw} and q_{vw} are at least $(1-\sqrt{\alpha})/n$. Hence,

$$r_{uv}^\sigma \;\geq\; [(1-2\alpha)n - 4\sqrt{n}] \cdot [(1-\sqrt{\alpha})/n]^2 \;\geq\; (1-3\sqrt{\alpha})/n.$$

□

We now present the SEPARATE algorithm, which distinguishes between strong and weak nodes. It uses a procedure ESTIMATE-RCP, which produces an estimate of the r_{uv} values. We will describe ESTIMATE-RCP momentarily.

SEPARATE
Input: Vertex $u \in V$.
Parameters: β, γ, δ, ℓ

1. Choose a random set R of $c \log n/\gamma$ vertices.
2. For every $v \in R$, run ESTIMATE-RCP(u, v), $c \log n$ times. If for a majority of these runs, ESTIMATE-RCP does not abort and outputs $\hat{r}_{uv} \geq (1-\beta)(1-\delta)/n$, then call v *accessible*.
3. If more than a $(1-2\gamma)$-fraction of vertices in R are accessible, then ACCEPT u. Otherwise REJECT v.

The main lemma of this section is about SEPARATE, which is proven using the previous lemma.

Lemma 3. *The procedure* SEPARATE *runs in $O(\sqrt{n}\ell \log^2 n)$ time, and with probability at least $1 - n^{-3}$ has the following behavior:*

1. *Assume there are at least $(1-\gamma)n$ strong vertices, for some constant γ. If u is strong, then the algorithm accepts u.*
2. *If u is weak, then the algorithm rejects u.*

Proof. For the first part, we assume that there are at least $(1-\gamma)n$ strong vertices. We now show that if u is a strong vertex, then this test will accept it with probability of error less than $1/n^3$. Since the sample size is $c \log n/\gamma$ (for sufficiently large c), a Chernoff bound argument shows that with probability of error less than $1/n^4$, at least a $(1-2\gamma)$ fraction of vertices in the sample must be strong.

Let v be a strong vertex in the sample. By Lemma 2, parts 2 and 3, we have $q_u(S_u^\sigma) \geq 1/2$, $q_v(S_v^\sigma) \geq 1/2$ and $r_{uv}^\sigma \geq (1-\beta)/n$. By Lemma 4, with probability at least 2/3, ESTIMATE-RCP will not abort and will output a value that is at least $(1-\beta)(1-\delta)/n$. Hence, with probability of error less than $1/n^4$, v will be deemed accessible. Taking a union bound over all errors, u is accepted with probability at least $1 - 1/n^3$.

Now we turn to the second part. Let u be a weak vertex. Then it is easy to see from the definition of weak vertices that there can be at most $7n/8$ vertices v with $Q_{uv} \geq 7/8n$. Suppose this is not the case. Let S be the set all the vertices v with $Q_{uv} < 1/n$. At least $7n/8$ have $Q_{uv} \geq 7/8n$. Thus,

$$\|Q_u - \vec{1}/n\|_{\text{TV}} \;\leq\; \sum_{v \in S}(1/n - Q_{uv}) \leq (7n/8)\cdot(1/8n) + (n/8)\cdot(1/n) \;<\; 1/4,$$

a contradiction. By the relation between q and Q probabilities, $r_{uv}^0 \leq Q_{uv}$, so again, there can be at most $7n/8$ vertices v with $r_{uv}^0 \geq 7/8n$. Now, in our random sample, (with probability of error less than $1/n^4$) a 1/9 fraction of vertices v must have $r_{uv}^0 \leq 7/8n$. Consider any such v. If ESTIMATE-RCP does not abort, then with probability at least 2/3, it will output an estimate of r_{uv}^0 that is at most $7/8n + \delta/2n$. By choosing constants α, β, δ to be sufficiently small, we can ensure that $7/8n + \delta/2n < (1-\beta)(1-\delta)/n$. Thus, with probability of error less than $1/n^4$, the algorithm will not call v accessible. A union bound over all v implies that with probability at least $1 - 1/n^3$, a 1/9 fraction of vertices in the random sample are not accessible from u, and if we choose $2\gamma < 1/9$, then the algorithm rejects u. □

We now describe the procedure ESTIMATE-RCP. It needs another procedure, FIND-SET, which given a vertex u, finds the set S_u^0 (approximately).

ESTIMATE-RCP
Input: Vertices $u, v \in V$.
Parameters: δ, $m = \sqrt{n}/\delta^2$.

1. Let $\hat{S}_u$:=FIND-SET(u), and $\hat{S}_v$:=FIND-SET(v).
2. Keep performing q-random walks from u until m such walks end at vertices in $\hat{S}_u$ (call this set of walks W_u). If more than $20m$ walks are performed, then ABORT.

   Do the same for walks starting from v as well to obtain the set of m walks W_v.
3. Let X be the number of pairwise collisions between walks in W_u and W_v (if a walk from W_u and a walk from W_v end at the same vertex, then this counts as a pairwise collision). Output X/m^2.

Lemma 4. *Let u and v be two vertices. The running time of* ESTIMATE-RCP *is $O(\sqrt{n}\ell \log n)$. If both $q_u(S_u^\sigma), q_v(S_v^\sigma) \geq 1/2$, then* ESTIMATE-RCP *aborts with*

probability less than $\exp(-O(m))$*. If* ESTIMATE-RCP *does not abort, then it outputs an estimate* $\tilde{r}_{uv}$ *such that with probability at least* $9/10$*,*

$$r^{\sigma}_{uv} - \delta \max\{r^{\sigma}_{uv}, 1/2n\} \;\leq\; \tilde{r}_{uv} \;\leq\; r^{0}_{uv} + \delta \max\{r^{0}_{uv}, 1/2n\}.$$

Proof. The running time bound is obvious, since the algorithm runs $O(\sqrt{n}\log n)$ random walks from u and v. By Lemma 6, with high probability, we have

$$S^{\sigma}_u \subseteq \hat{S}_u \subseteq S^{0}_u \quad \text{and} \quad S^{\sigma}_v \subseteq \hat{S}_v \subseteq S^{0}_v.$$

Thus, if $q_u(S^{\sigma}_u) \geq 1/2$, by a Chernoff bound, the probability that ESTIMATE-RCP aborts while performing walks from u is less than $\exp(-O(m))$. This proves the first part.

Define $\hat{r}_{uv}$ as:

$$\hat{r}_{uv} = \sum_{w \in \hat{S}_u \cap \hat{S}_v} q_{uw} q_{vw}$$

Since (with high probability), $S^{\sigma}_u \cap S^{\sigma}_v \subseteq \hat{S}_u \cap \hat{S}_v \subseteq S^{0}_u \cap S^{0}_v$, we get that $r^{\sigma}_{uv} \leq \hat{r}_{uv} \leq r^{0}_{uv}$. Assuming that the algorithm does not ABORT, let X be the random variable denoting number of pairs of walks that collide. The algorithm's estimate is X/M, where $M = m^2$. We now show that this is a good estimate, with a high probability. For any pair of walks, the probability that they collide is $\sum_{w \in \hat{S}_u \cap \hat{S}_v} q_{uw} \cdot q_{vw} = \hat{r}_{uv}$. Thus, the expectation of X is exactly $\hat{r}_{uv} M$. We can show that $\mathbf{Var}(X) \leq 4\hat{r}_{uv} M^{3/2}/\sqrt{n}$ (see Lemma 5). Thus, by Chebyshev's inequality, if $b = \max\{\hat{r}_{uv}, 1/2n\}$, then

$$\mathbf{Pr}[|X - \hat{r}_{uv} M| > (\delta b) M] \;\leq\; \frac{4\hat{r}_{uv} M^{3/2}}{\sqrt{n}[(\delta b) M]^2} \;\leq\; 1/10$$

since $M = n/\delta^4$, and we can choose δ to be a small enough constant. This implies that with probability at least $9/10$,

$$r^{\sigma}_{uv} - \delta \max\{r^{\sigma}_{uv}, 1/2n\} \;\leq\; \tilde{r}_{uv} \;\leq\; r^{0}_{uv} + \delta \max\{r^{0}_{uv}, 1/2n\}.$$

□

Lemma 5. *The variance of* X*, the number of pairwise collisions when* ESTIMATE-RCP *is run on vertices* u *and* v*, is bounded by* $\mathbf{Var}(X) \leq 4\hat{r}_{uv} M^{3/2}/\sqrt{n}$*.*

Proof. For convenience of notation, let $R := \hat{r}_{uv}$. We index the walks from u by $i = 1, 2, \ldots, m$, and the walks from v by $j = 1, 2, \ldots, m$. Define indicator random variables X_{ij} for any $i, j \in \{1, 2, \ldots, m\}$ which are set to 1 if the i^{th} walk from u and the j^{th} walk from v collide. Note that $\mathbf{Pr}[X_{ij} = 1] = R$. The random variable X equals $\sum_{ij} X_{ij}$, and thus the expectation of X is $Rm^2 = RM$. We now estimate the variance of X as follows:

$$\mathbb{E}[X^2] = \sum_{ij} E[X_{ij}^2] + \sum_{i \neq i', j \neq j'} E[X_{ij} X_{i'j'}] + \sum_{i, j \neq j'} E[X_{ij} X_{ij'}] + \sum_{j, i \neq i'} E[X_{ij} X_{i'j}].$$

We estimate each term separately. First, $E[X_{ij}^2] = R$. Then, if $i \neq i'$ and $j \neq j'$, then X_{ij} and $X_{i'j'}$ are independent, and hence $E[X_{ij}X_{i'j'}] = R^2$.

The last case is if $i = i'$ and $j \neq j'$ (the case when $i \neq i'$, $j = j'$ is handled analogously). Then, we have

$$\begin{aligned} E[X_{ij}X_{ij'}] &= \mathbf{Pr}[X_{ij}X_{ij'} = 1] \\ &= \sum_{w \in \hat{S}_u \cap \hat{S}_v} q_{uw} q_{vw}^2 \\ &\leq \sum_{w \in \hat{S}_u \cap \hat{S}_v} \frac{1}{\sqrt{n}} \cdot q_{uw} q_{vw} \\ &= \frac{1}{\sqrt{n}} \cdot R. \end{aligned}$$

Here, we crucially use the fact that for all vertices $w \in \hat{S}_v$, $q_{vw} \leq 1/\sqrt{n}$. This is what bounds the variance and allows us to give a sublinear time procedure to estimate r_{uv} values.

Thus, we have

$$\begin{aligned} \mathbf{Var}[X] &= E[X^2] - E[X]^2 \\ &\leq m^2 R + m^2(m-1)^2 R^2 + 2m^2(m-1) \cdot \frac{R}{\sqrt{n}} - m^4 R^2 \\ &\leq 4m^3 \cdot \frac{R}{\sqrt{n}} && \because m = \Omega(\sqrt{n}) \\ &= 4RM^{3/2}/\sqrt{n}. \end{aligned}$$

□

Now we describe the procedure Find-Set. Since we cannot get S_u^0 (or indeed any such set) exactly, we have to settle for some errors. Our final procedure will not exactly estimate r_{uv}^0 but some closely related value. Note that what we actually require of Find-Set is just an oracle that given a vertex v, tells us whether or not it is in S_u^σ. Since S_u^σ is of linear size, the procedure actually outputs the complement (which is of size $O(\sqrt{n})$).

Find-Set
Input: Vertex $u \in V$.

1. Perform $c\sqrt{n}\log n$ independent q-random walks of length ℓ from u.
2. Return $\hat{S}_u$, the set of all vertices w such that at most $c(1-\sigma/2)\log n$ q-random walks from u end at w.

Lemma 6. *With probability at least $1 - 1/n$, the procedure* Find-Set*, on input vertex u, outputs a set $\hat{S}_u$ such that*

$$S_u^\sigma \subseteq \hat{S}_u \subseteq S_u^0.$$

Proof. Any vertex $w \in S_u^\sigma$ has $q_{uw} \leq (1-\sigma)/\sqrt{n}$, and any vertex $v \in V \setminus S_u^0$ has $q_{uv} > 1/\sqrt{n}$. Since we run $c\sqrt{n}\log n$ random walks from u, the expected number of times such a walk hits w is at most $c(1-\sigma)\log n$, and the expected number of times such a walk hits v is at least $c\log n$. Thus, by a Chernoff bound, the chance we have more than $c(1-\sigma/2)\log n$ walks hitting w and less than $c(1-\sigma/2)\log n$ walks hitting v is at most $1/n^2$, by making c large enough. The statement of the lemma follows by a union bound. □

4.3.2 Hybridizing the graph with an expander

Now that we have a separating procedure, we can describe the actual reconstruction procedure RECONSTRUCT that hybridizes an expander with our original graph. Given a query vertex v, the procedure will output at most d vertices which will be the neighbors of v in the reconstructed graph. We will refer to the final reconstructed graph as G', and for cuts $(S, \bar{S})$ in G', we use the notation $E'(S, \bar{S})$ to refer to the set of edges crossing the cut.

Since we are describing a parallel filter, we assume we have access to a sublinear sized random seed s of size $\widetilde{O}(\sqrt{n})$ which is fixed for all queries. Since the total number of calls to SEPARATE is at most $O(n)$ (for each call to the RECONSTRUCT, we will make $O(1)$ calls to SEPARATE), by taking a union bound over all the error probabilities, we can ensure that the guarantees of Lemma 3 hold with probability at least $1 - 1/n^2$. Since the seed is fixed, we can unambiguously refer to vertices as *accepted* or *rejected*, based on SEPARATE.

For constructing G', we use an explicit bounded degree expander G^* with n vertices. The explicit construction allows us to find all neighbors of a vertex $v \in G^*$ in $\text{poly}(\log n)$ time. This expander G^* has degree bound $d/2$ and expansion at least ηd for some constant η. Naturally, there is a one-to-one correspondence between the vertices of G and G^*. Abusing notation, given a vertex v in G, we will refer to the corresponding vertex in G^* also as v. To prevent confusion about edges, we will call edges G, G^*, or G'-edges depending on the graph in consideration.

For the sake of intuition, we can think of the accepted vertices as strong, and the rejected as weak. The reconstructed graph G' will be a careful combination of G and G^*. Starting with G, here is an informal description of how G' is built. For any rejected vertex v, we remove all G-edges incident to v and add all G^*-edges incident to v to get the G^*-edges. This is to ensure that any subset of rejected vertices will have large conductance. For accepted vertices, we would like to just keep the same G-edges.

This construction could potentially make some (accepted) vertices have a degree larger than d. Therefore, we have to remove some G-edges incident to accepted vertices to maintain the degree bound. These edges are removed based on a simple *local* rule. Note that this choice cannot be arbitrary, since we want a parallel filter (for example, if the G-edge (u, v) is removed on querying v, then it must also be removed on querying u). Unfortunately, this might affect the condutance of subsets of accepted vertices. We ensure that every time we remove such an edge, we replace it by a very short path (again decided by local considerations) between the endpoints

without affecting the degree bound. This gives us G' with the desired properties. Because we want every query to be handled in sublinear time, we give a procedure that determines the neighbors of a vertex in G' by running SEPARATE on a constant number of vertices. We now state the main result of this section.

Lemma 7. *The procedure* RECONSTRUCT *runs in* $O(\sqrt{n}\ell \log^2 n)$ *time and needs a random seed* s *of* $O(\sqrt{n}\ell \log^2 n)$ *bits. Its outputs are consistent over all queries (i.e. vertex* v *is output as a neighbor of* u *(in* G'*) iff* u *is output as a neighbor of* v*). The final graph* G' *has degree bound* d*. Furthermore, any cut in* G' *has higher conductance than the same cut in* G*, and any weak vertex has all its* G^**-neighbors adjacent in* G'*.*

We assume that the explicit expander G^* is a strong edge expander with the following property:

Property 4.3.1. *The expander* G^* *has degree bound* $d/2$*. For any set of vertices* S *in* G^* *with* $|S| \leq n/2$*, we have* $|E^*(S, \bar{S})| \geq \eta|S|d$*, where* η *is a constant.*

We assume that there is some global ordering of the vertices (say, according to the value of their indices). Thus, given any vertex v, there is an *ordered* list of the neighbors of v, with possibly some "null" entries at the end (because v might have degree less than d). When we refer to the i^{th} vertex in some list of vertices, we mean the i^{th} vertex in the list in the global ordering.

As mentioned before, if any vertex u is rejected by SEPARATE, we remove all the G-edges incident on it and replace them by G^*-edges. To avoid increasing the degree of accepted vertices, we need to remove some G-edges between accepted vertices as well. We now describe a procedure that given an edge $e = (u, v)$ of G, where u and v are both accepted by SEPARATE, outputs whether the edge needs to be removed, and if so, what edges are added (because of this removal). The procedure REMOVE involves $O(d)$ calls to SEPARATE.

REMOVE
Input: Edge $(u, v) \in G$.

1. Find the G^*-neighbors of u and v and consider all the rejected vertices (according to SEPARATE).
2. Suppose v is the i^{th} neighbor of u, and u is the j^{th} neighbor of v. Let s be i^{th} rejected G^*-neighbor of u, and let t be the j^{th} rejected G^*-neighbor of v. Note that one or both of s and t could be null.
3. If both s and t are null, then (u, v) is not removed. If t is null, then the edges (s, u) and (s, v) replace (u, v). If s is null, then the edges (t, u) and (t, v) replace (u, v). If neither s nor t is null, then the edges $(u, s), (s, t), (t, v)$ replace (u, v). In any case, output the new neighbors to u and v.

Note that if an edge (u, v) is replaced, it is replaced by a path from u to v. Furthermore, for two different edges, these paths are edge-disjoint. This gives the following claim.

Claim 4.3.2. $|E'(S, \bar{S})| \geq |E(S, \bar{S})|$

Using the procedure REMOVE, we can describe the main reconstruction procedure. This procedure will output all the neighbors of an input vertex u on the reconstructed graph and involves calling REMOVE on $O(d)$ edges.

RECONSTRUCT
Input: Vertex $v \in V$.

1. Using SEPARATE, check if v is accepted or rejected.
2. If v is accepted : Call REMOVE on all the G-edges incident on v. All these outputs give the neighbors of v in the reconstructed graph.
3. If v is rejected : Remove all G-edges incident on v. Find the G^*-neighbors of v. For each such u:
   (a) If u is rejected, then add edge (u, v) to G'.
   (b) If not, suppose v is the i^{th} rejected G^*-neighbors of u. Let w be the i^{th} G-neighbor of u. If w is rejected by SEPARATE, then add (u, v). Otherwise, call REMOVE on (u, w).

   All these calls together will output the neighbors of v.

Claim 4.3.3. *The procedure* RECONSTRUCT *is consistent: vertex v is output as a neighbor of u (in G') iff u is output as a neighbor of v.*

Proof. Suppose u is an accepted vertex and let v be output as a G'-neighbor of u. The neighbor v came about because of a call to REMOVE on a G-edge (u, w). If (u, w) is not removed, then w and v are the same. In this case, u will also be output as a neighbor of v in G'. Suppose (u, w) is replaced by (u, v) and (w, v) for some rejected vertex v. Then, for some i, v is the i^{th} rejected G^*-neighbor of u (or w), and w is the i^{th} G-neighbor of u (or vice versa). When RECONSTRUCT is called on v, then we can see that REMOVE will be called on (u, w), and u will be output as a G'-neighbor of v.

Now suppose that (u, w) is replaced by $(u, s), (s, t), (t, w)$, where s, t are rejected vertices. Using a similar argument as the one above, we can show that u will be output as a G'-neighbor of v.

Let u be a rejected vertex. If v is a rejected vertex that is a G'-neighbor of u, then u will also be output as a G'-neighbor of v. If v is accepted, then we can make the above argument to prove consistency. □

4.4 Bounds on Number of Edges Changed and Conductance

We now bound the number of edges changed by RECONSTRUCT in terms of the optimal number of edges to be changed to make the conductance at least ϕ. Theorem 3 gives such a bound and thus establishes a link between two notions of measuring the "distance" to having a large conductance: number of edges that needed to be changed, and the number of vertices from which the random walk mixes poorly (i.e. the weak vertices).

Theorem 3. *The reconstruction algorithm achieves an approximation ratio of $O(1/\phi)$ to the optimal number of edges to be changed to make the conductance of the graph at least ϕ.*

This follows from the following lemma that shows that there is a large cut of low conductance:

Lemma 8. *Let S be the set of strong vertices. Then there is a cut $(B, V\setminus B)$ with the property that $\Omega(n - |S|) \leq \min\{|B|, |V \setminus B|\}$, which has conductance less than $\phi/2$.*

Proof. We keep recursively partitioning the graph, finding cuts in the remaining induced graph of conductance less than $\phi/2$, and aggregating these cuts. Our goal is to show that the final cut obtained has the properties required here.

To be more precise: start out with $B = \{\}$. Let $\bar{B} = V \setminus B$. If there is a cut $(S, \bar{S})$ in $\bar{B}$ with $|S| \leq |\bar{B}|/2$ having conductance less than $\phi/2$, then we set $B := B \cup S$, and continue, as long as $|B| \leq n/2$. If $|B|$ exceeds $\frac{\alpha}{2}n$, then note that $|B| \leq (\frac{1}{2} + \frac{\alpha}{4})n$, and we are done since $\min\{|B|, |V \setminus B|\} \geq \Omega(n) \geq \Omega(n - |S|)$.

We therefore assume that $|B| \leq \frac{\alpha}{2}n$. It is easy to check that the final cut $(B, \bar{B})$ also has conductance less than $\phi/2$. Furthermore, the subgraph induced on $\bar{B}$ has conductance at least $\phi/2$, or in other words, the $\bar{B}$-conductance of G is at least $\phi/2$.

Now, we apply Theorem 2 to G, to conclude that there is a set B' such that $|B'| \leq 2|B|$ (because the stationary distribution is uniform) with the property that starting from any $s \in V \setminus B'$, the uniform averaging walk, after ℓ steps, ends up in a distribution $\bar{p}_s^\ell = q_s$ such that

$$\|q_s - \vec{1}/n\|_{\mathrm{TV}} \;\leq\; \alpha/2 + |B|/n, \tag{4.2}$$

if we choose $\ell \geq 64\log(2n/\alpha)/(\alpha\phi^2)$. Since $|B| \leq \frac{\alpha}{2}n$, then inequality (4.2) implies that we have $\|q_s - \vec{1}/n\|_{\mathrm{TV}} \leq \alpha$ for all $s \in V \setminus B'$. Thus, all vertices $s \in V \setminus B'$ are strong. Hence, $|S| \geq n - |B'| \geq n - 2|B|$, and so $\min\{|B|, |V \setminus B|\} = |B| \geq \Omega(n - |S|)$ in this case as well. □

Proof. (Theorem 3) Lemma 8 immediately implies that the optimal reconstruction of the graph must add at least $\Omega(\phi d(n - |S|))$ edges to patch up the cut $(B, V \setminus B)$. Whenever the reconstruction algorithm adds an edge, then one of the endpoints is a rejected vertex. The total number of removed edges is at most the number of added

edges. Therefore, we can bound the total change (up to constant factors) by d times the number of rejected vertices. Suppose the number of strong vertices is less than $(1-\gamma)n$. The graph G is trivially changed by atmost $O(nd) = O(d(n-|S|))$ edges. If the number of strong vertices is more than $(1-\gamma)n$, then by Lemma 3 all strong vertices are accepted. Our reconstruction algorithm potentially adds $O(d(n-|S|))$ edges, which means that we have an approximation ratio of $O(1/\phi)$ to the optimal number of edges to be changed. □

Now we give bounds on the conductance of G'.

Theorem 4. *The reconstructed graph G' has conductance at least $\Omega(\phi^2/\log n)$.*

Proof. Consider a cut $(S, \bar{S})$ in the reconstructed graph G', with $|S| \leq n/2$. Suppose S has a subset T of more than $(1-\eta/2)|S|$ weak vertices (where η comes from Property 4.3.1). By the properties in Lemma 7, all the edges incident to these weak vertices in G^* are present in G'. The number of edges leaving T is at least $\eta|T|d$, whereas the number of edges incident on $S \backslash T$ is at most $\eta|T|d/2$. This implies that the conductance of S is $\Omega(1)$.

So assume that less than a $(1-\eta/2)$-fraction of the vertices in S are weak. Then we claim that the conductance of S is already at least $\eta/16\ell$. By Lemma 7, it suffices to prove this for G. Assume for the sake of contradiction that the conductance of S (in G) is less than $\eta/16\ell$. Then the following modification of the proof of Lemma 4.7 in the work Czumaj and Sohler [35] gives us the desired contradiction.

Consider a Q-random walk starting from the uniform distribution on the vertices. Since the uniform distribution is stationary, for any $k \geq 0$, the k^{th} vertex in the walk is uniformly distributed in V. Thus, for any edge that is not a self-loop, the chance that it is selected in the next step of the random walk is $(1/n) \cdot (1/2d) \cdot 2 = 1/nd$. Thus, we have

$$\mathbf{Pr}[\text{an edge in } E(S,\bar{S}) \text{ is used in the step } k+1] \;\leq\; \frac{E(S,\bar{S})}{nd}.$$

Since the walk runs for at most 2ℓ steps, the chance of using an edge in $E(S, \bar{S})$ is at most $\frac{2\ell E(S,\bar{S})}{nd}$. The chance that the walk starts in S is $\frac{|S|}{n}$. Since the conductance of the cut $E(S,\bar{S})/d|S| < \eta/16\ell$, we get that

$$\mathbf{Pr}[\text{walk starts in } S \text{ and doesn't use an edge in } E(S,\bar{S})] \;\geq\; (1-\eta/8)\frac{|S|}{n}.$$

Thus, conditioned on starting in S, the chance that the walk never crosses the cut (or in other words, never leaves the set S) is at least $(1-\eta/8)$.

Therefore, for at least a $(1-\eta/2)$-fraction of vertices in S, the chance that the random walk starting from them never leaves the set S is at least $3/4$. On the set $\bar{S}$, the uniform distribution places a mass of at least $1/2$ (since $|\bar{S}| \geq 1/2$), while the Q-random walk from one of the aforementioned vertices $u \in S$ places a mass of at most $1/4$, which implies that $\|q_u - \frac{\vec{1}}{n}\|_{\text{TV}} \geq 1/4$. This implies that u is weak, and thus S contains at least a $(1-\eta/2)$-fraction of weak vertices, a contradiction. □

Chapter 5

Self-improving Algorithms

We design self-improving algorithms for sorting and computing Delaunay triangulations. As discussed in the Section 1.2, we assume that the inputs (either lists of numbers, or lists of points) are repeatedly coming from a fixed but unknown distribution. A self-improving algorithm starts learning about this distribution and tries to optimize its running time based on this information. We first explain the self-improving sorter, as a sort of primer. Then, we show how to extend these techniques to the geometric realm for computing Delaunay triangulations.

5.1 A Self-Improving Sorter

The self-improving sorter takes an input $I = (x_1, x_2, \cdots, x_n)$ of numbers drawn from a distribution $\mathcal{D} = \prod_i \mathcal{D}_i$ (ie, each x_i is chosen independently from $\mathcal{D}_i$). We denote by $\mathcal{D}_<$ the distribution over the symmetric group induced by the ranks of the x_i's (using the indices i to break ties). By an information theoretic argument, it is easy to see that any sorter must take expected $\Omega(H(\mathcal{D}_<) + n)$ comparisons with respect to $\mathcal{D}$. This is, indeed, the bound that our self-improving sorter achieves.

For simplicity, we begin with the steady-state algorithm and discuss the training phase later. We also assume that the distribution $\mathcal{D}$ is known ahead of time and that we are allowed some amount of preprocessing before having to deal with the first input instance (§5.1.1). Both assumptions are unrealistic, so we show how to remove them to produce a bona fide self-improving sorter (§5.1.2). The surprise is how strikingly little of the distribution needs to be learned for effective self-improvement.

Theorem 5.1.1. *There exists a self-improving sorter of $O(H(\mathcal{D}_<) + n)$ limiting complexity, for any input distribution $\mathcal{D} = \prod_i \mathcal{D}_i$. Its worst case running time is $O(n \log n)$. If the input $(x_1, \ldots, x_n)$ to be sorted is obtained by drawing each x_i independently (from a distribution that might depend on i), then for any $\varepsilon > 0$ the storage can be made $O(n^{1+\varepsilon})$ for an expected running time of $O(\varepsilon^{-1} H(\mathcal{D}_<) + n)$: this tradeoff is optimal for distributions of high enough entropy. The algorithm reaches its steady state within $O(n^{\varepsilon} \log n)$ rounds.*

REMARK: Much research has been done on adaptive sorting [42], especially on algorithms that exploit near-sortedness. Our approach is conceptually different. As we mentioned in the previous section, we seek to exploit properties, not of individual inputs, but of their distribution. In particular, our sorter runs in linear time for permutations drawn from a linear-entropy source, even though any individual input might be a *perfectly random* permutation. We are not aware of any previous algorithm that can achieve that.

Can we hope for a similar result similar to Theorem 5.1.1 if we drop the independence assumption? The short answer is no.

Lemma 5.1.2. *There exists an input distribution $\mathcal{D}$ such that any comparison-based algorithm that can sort a random input from $\mathcal{D}$ in expected $O(H(\mathcal{D}_<) + n)$ time requires at least $\Omega(2^{H(\mathcal{D}_<)} n \log n)$ storage. This holds for any value of the entropy $H(\mathcal{D}_<)$ that is smaller than $n \log n$ by a large enough constant factor.*

Proof. Consider the set of all $n!$ permutations. Every subset S of 2^h permutations induces a distribution $\mathcal{D}^S$ defined by picking every permutation in S with equal probability and none other. Note that the total number of distributions is $\binom{n!}{2^h} > (n!/2^h)^{2^h}$ and $H(\mathcal{D}^S_<) = h$. Suppose there exists a comparison-based algorithm $\mathcal{A}_S$ that sorts a random input from $\mathcal{D}^S$ in expected time at most $c(n + h)$, for some constant $c > 0$. By Markov's inequality this implies that at least half of the permutations of S are sorted by $\mathcal{A}_S$ in at most $2c(n + h)$ comparisons. But, within $2c(n+h)$ comparisons, the algorithm $\mathcal{A}_S$ can sort a set P of at most $2^{2c(n+h)}$ permutations. Therefore, any other S' such that $\mathcal{A}_{S'} = \mathcal{A}_S$ will have to draw at least half of its elements from P. This limits the number of such S' to

$$\binom{n!}{2^h/2}\binom{2^{2c(n+h)}}{2^h/2} < (n!)^{2^{h-1}} 2^{c(n+h)2^h}.$$

This means that the number of distinct algorithms needed exceeds

$$(n!/2^h)^{2^h}/((n!)^{2^{h-1}} 2^{c(n+h)2^h}) > (n!)^{2^{h-1}} 2^{-(c+1)(n+h)2^h} = 2^{\Omega(2^h n \log n)},$$

assuming that $h/(n \log n)$ is small enough. An algorithm is entirely specified by a string of bits; therefore at least one such algorithm must require storage logarithmic in the previous bound. □

□

Fredman [44] gives a comparison-based algorithm that can optimally sort any distribution of permutations, but uses an exponentially large data structure to decide which comparisons to perform. This result shows that the storage used by Fredman's algorithm is essentially optimal.

5.1.1 Sorting with Full Knowledge

We consider the problem of sorting $I = (x_1, \ldots, x_n)$, where each x_i is drawn from a distribution $\mathcal{D}_i$, which is specified by a vector $(p_{i,1}, \ldots, p_{i,N})$, where $p_{i,j} = \text{Prob}[\, x_i =$

$j\,]$ [1]. We can assume without loss of generality that all the x_i's are distinct. (If not, simply replace x_i by $nx_i + i - 1$ for tie-breaking purposes and enlarge N to $n(N+1)$. All probabilities and entropies remain the same.)

The first step of the self-improving sorter is to sample $\mathcal{D}$ a few times (the training phase) and create a "typical" instance to divide the real line into a set of disjoint, sorted intervals. Next, given some input I, the algorithm sorts I by using the typical instance, placing each input number in its respective interval. All numbers falling into the same intervals are then sorted in a standard fashion. The algorithm needs a few supporting data structures.

- THE V-LIST: Fix an integer parameter $\lambda = c \log n$, for large enough c, and sample λ input instances from $\prod \mathcal{D}_i$. Form their union and sort the resulting λn-element multiset into a single list $u_1 \leq \cdots \leq u_{\lambda n}$. Next, extract from it every λ-th item and form the list $V = (v_0, \ldots, v_{n+1})$, where $v_0 = 0$, $v_{n+1} = \infty$, and $v_i = u_{i\lambda}$ for $0 < i \leq n$. Keep the *V-list* in a sorted table as a snapshot of a "typical" input instance. We will prove the remarkable fact that, with high probability, locating each x_i in the V-list is linearly equivalent to sorting I. We cannot afford to search the V-list directly, however. To do that, we need auxiliary search structures.

- THE D_i-TREES: For any $i > 0$, let pred_i^V be the predecessor[2] of a random y from $\mathcal{D}_i$ in the V-list, and let H_i^V be the entropy of pred_i^V (which cannot exceed the entropy of $\mathcal{D}_i$). The D_i-tree is an optimum binary search tree [62] over the keys of the V-list, where the access probability of v_k is $\sum_j \{\, p_{i,j} \mid v_k \leq j < v_{k+1} \,\}$ [3], for any $0 \leq k \leq n$: the same distribution used to define H_i^V. This allows us to compute pred_i^V using $O(H_i^V + 1)$ expected comparisons.

The total space used is $O(n^2)$. This can be decreased to $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$; we describe how later. As we explained earlier, the input I is sorted by a two-phase procedure. First we search for each x_i in the V-list using the previous technique. This allows us to partition I into groups $G_1 < G_2 < \cdots$ of x_i's sharing the same predecessor in the V-list. The next phase involves going through each G_j and sorting their elements naively, say using insertion sort. The first phase of the algorithm takes $O(n + \sum_i H_i^V)$ expected time.[4] What about the second? Its complexity is $O(n)$, as follows from:

Lemma 5.1.3. *With probability $> 1 - n^{-2}$ over the construction of the V-list -*

$$\mathbf{E}_{\mathcal{D}}\big[\,|\,\{\, i \mid v_k \leq x_i < v_{k+1} \,\}\,|^2\big] = O(1), \textit{for all } 0 \leq k \leq n.$$

[1] All the arguments we give shall hold directly (and obviously) even if the $\mathcal{D}_i$'s are continuous. We have made this assumption for ease of presentation.

[2] Throughout this paper, the predecessor of y in a list refers to the index of the largest list element $\leq y$; it does not refer to the element itself.

[3] If the $\mathcal{D}_i$'s were continuous, then this would be defined as the probability of x_j falling in $[v_k, v_{k+1})$.

[4] The H_i^V's themselves are random variables depending on the choice of the V-list. Therefore, this is a conditional expectation.

Proof. Remember that the V-list was formed by taking certain elements from a list $u_1 \leq \cdots \leq u_{\lambda n}$, where $\lambda = c \log n$. Consider two points u_i and u_j. Note that all the other $\lambda n - 2$ points are independent of these two points. For every $\ell \notin \{i, j\}$, let $X_\ell^{(t)}$ be the indicator random variable for the event that $u_\ell \in [u_i, u_j) = t$. Let $X^{(t)} = \sum_\ell X_\ell^{(t)}$. Since all the $X_\ell^{(t)}$'s are independent, by Chernoff's bound [11], for any $\beta \in (0, 1]$ -

$$\Pr[X^{(t)} \leq (1-\beta)\mathbf{E}[X^{(t)}]] \leq e^{-\beta^2 \mathbf{E}[X^{(t)}]/2}$$

With probability at least $1 - n^{-4}$, if $\mathbf{E}[X^{(t)}] > 4c \log n$, then $X^{(t)} > 2c \log n$. We can apply the same argument for any pair u_i, u_j. Taking a union bound over all pairs, we get that with probability $> 1 - n^{-2}$, if for the pair t, $\mathbf{E}[X^{(t)}] > 4c \log n$, then $X^{(t)} > 2c \log n$.

The V-list is constructed such that for $t_k = [v_k, v_{k+1})$, $X^{(t_k)} \leq c \log n$. Let $Y_i^{(t_k)}$ be the indicator random variable for the event that $x_i \in_R \mathcal{D}_i$ lies in t_k, and $Y^{(t_k)} = \sum_i Y_i^{(t_k)} = |\{ i \,|\, v_k \leq x_i < v_{k+1} \}|$. Note that $\mathbf{E}[X^{(t_k)}] \geq (\log n - 2)\mathbf{E}[Y^{(t_k)}]$ and therefore, $\mathbf{E}[Y^{(t_k)}] = O(1)$. By independence of the $\mathcal{D}_i$'s and by linearity of expectation -

$$\begin{aligned} \mathbf{E}[{Y^{(t_k)}}^2] &= \mathbf{E}[(\sum_i Y_i^{(t_k)})^2] = \sum_i \mathbf{E}[{Y_i^{(t_k)}}^2] + 2\sum_{i<j} \mathbf{E}[Y_i^{(t_k)}]\mathbf{E}[Y_j^{(t_k)}] \\ &\leq \sum_i \mathbf{E}[Y_i^{(t_k)}] + (\sum_i \mathbf{E}[Y_i^{(t_k)}])^2 = O(1) \end{aligned}$$

□

We have shown that the algorithm takes $O(n + \sum_i H_i^V)$ time (given a fixed V-list) plus an $O(n)$ additive expected term (over V and $\mathcal{D}$). We now show that this running time is indeed optimal.

Lemma 5.1.4.

$$\sum_i H_i^V = O(n + H(\mathcal{D}_<))$$

We will actually show this to be the case for *any* linear sized sorted list V. We will need a basic claim, which shall be proven for completeness, about the joint entropy of independent random variables.

Claim 5.1.5. *Let $H(pred_1^V, pred_2^V, \cdots, pred_n^V)$ be the joint entropy of the random variables*
$pred_1^V, pred_2^V, \cdots, pred_n^V$. Then -

$$H(pred_1^V, pred_2^V, \cdots, pred_n^V) = \sum_i H_i^V$$

Proof. This is a consequence of the independence of the $\mathcal{D}_i$'s. We will prove this by induction over the number of variables the joint entropy includes. For the base case,

$H(pred_1^V) = H_1^V$ by definition. Assume inductively that $H(pred_1^V, pred_2^V, \cdots, pred_k^V) = \sum_{i=1}^{k} H(pred_i^V)$. By the chain rule for conditional entropy [5] and independence,

$$\begin{aligned} & H(pred_1^V, pred_2^V, \cdots, pred_{k+1}^V) \\ = \; & H(pred_1^V, pred_2^V, \cdots, pred_k^V | pred_{k+1}^V) + H(pred_{k+1}^V) = \sum_{i=1}^{k+1} H_i^V \end{aligned}$$

□

It suffices to prove the following lemma.

Lemma 5.1.6.

$$H(pred_1^V, pred_2^V, \cdots, pred_n^V) = O(n + H(\mathcal{D}_<))$$

Proof. Let there be a string representation $s(\pi)$ for all permutations π. Let $\pi(I)$ denote the permutation induced by the input I. By information theory, there exists a string encoding s such that $\mathbf{E}_{\mathcal{D}}[|s(\pi(I))|] = O(H(\mathcal{D}_<))$. Given $s(\pi(I))$, we can uniquely identify the permutation induced by $\pi(I)$. If we know $\pi(I)$, we can in linear time merge I with V to get a sorted list. Then, in linear time, we can output $pred_1^V, pred_2^V, \cdots, pred_n^V$. Therefore, the output $pred_1^V, pred_2^V, \cdots, pred_n^V$ can be uniquely identified by $s(\pi(I))$ and $2cn$ more bits. Again, by information theory, $\mathbf{E}[|s(\pi(I)) + 2cn] \geq H(pred_1^V, pred_2^V, \cdots, pred_n^V)$. □

This completes the proof for the optimality of the time taken by the sorter. We now show that the storage can be reduced to $O(n^{1+\varepsilon})$, for any $\varepsilon > 0$. The main idea is to prune each of the D_i trees to depth $\varepsilon \log n$. This ensures that each of these trees has size $O(n^{\varepsilon})$ and the total storage used is $O(n^{1+\varepsilon})$. We also construct a completely balanced binary tree T for searching in the V-list. Now, when we wish to search for x_i in the V-list, we first search using the pruned D_i-tree. At the end, if we reach a leaf of the *unpruned* D_i-tree, we stop since we have found the right interval of the V-list which contains x_i. On the other hand, if the search in the D_i-tree was unsuccessful, then we use T for searching.

In the first case, the time taken for searching is simply the same that it would have taken with unpruned D_i-trees. In the second case, the time taken is $O((1 + \varepsilon) \log n)$. But note that the time taken with unpruned D_i-trees is $> \varepsilon \log n$ (since the search on the pruned D_i-tree failed, we must have reached some internal node of the unpruned tree). Therefore, the extra time taken is only a $O(\varepsilon^{-1})$ factor of the original time. As a result, the space can be reduced to $O(n^{1+\varepsilon})$ with only a constant factor increase in running time (for any fixed $\varepsilon > 0$).

We can show that the storage cannot be reduced to linear. In fact, the tradeoff between the $O(n^{1+\varepsilon})$ storage bound and an expected running time off the optimal by a factor of $O(1/\varepsilon)$ is optimal.

[5] Given two random variables X and Y over supports $\mathcal{X}$ and $\mathcal{Y}$, the conditional entropy $H(Y|X) = \sum_{x \in \mathcal{X}} \Pr(X = x) H(Y|X = x)$. The chain rule tells us that $H(Y, X) = H(Y|X) + H(X)$

Lemma 5.1.7. *For any c large enough and any $h \leq \frac{3}{c} n \log n$, there is a distribution $\mathcal{D} = \prod_i \mathcal{D}_i$ of entropy h such that any comparison-based algorithm that can sort a random permutation from $\mathcal{D}$ in expected time $c(h+n)$ requires a data structure of bit size $\Omega(2^{h/n} n \log n)$.*

Proof. The proof is a specialization of the argument used for proving Lemma 5.1.2. Let $\kappa = 2^{\lfloor h/n \rfloor}$. We define $\mathcal{D}_i$ by choosing κ distinct integers in $[1, n]$ and making them equally likely to be picked as x_i. This leads to $\binom{n}{\kappa}^n > (n/\kappa)^{\kappa n}$ choices of distinct distributions $\mathcal{D}$. Suppose that there is a data structure of size s that can accommodate any such distribution with an expected running time of at most $c(h+n)$. Then one such data structure $\mathcal{S}$ must be able to accommodate this running time for a set $\mathcal{G}$ of at least $(n/\kappa)^{\kappa n} 2^{-s}$ distributions $\mathcal{D}$. Any input instance that is sorted in at most $2c(h+n)$ time by this data structure is called *easy*: the set of easy instances is denoted by $\mathcal{E}$.

Each $\mathcal{D}_i$ is characterized by a vector $v_i = (a_{i,1}, \ldots, a_{i,\kappa})$, so that $\mathcal{D}$ itself is specified by $v = (v_1, \ldots, v_n) \in \mathbf{R}^{n\kappa}$. (From now on, we view v both as a vector and a distribution of input instances.) Define the j-th projection of v as $v^j = (a_{1,j}, \ldots, a_{n,j})$. Even if $v \in \mathcal{G}$, it could well be that none of the projections of v are easy. However, if we consider the projections obtained by permuting the coordinates of each vector $v_i = (a_{i,1}, \ldots, a_{i,\kappa})$ in all possible ways we enumerate each input instance from v the same number of times. Note that applying these permutations gives us different vectors which also represent $\mathcal{D}$. Since the expected time to sort an input chosen from $\mathcal{D} \in \mathcal{G}$ is at most $c(h+n)$, by Markov's inequality, there exists a choice of permutations (one for each $1 \leq i \leq n$) for which at least half of the projections of the vector obtained by applying these permutations are easy.

Let us count how many distributions have a vector representation with a choice of permutations placing half its projections in $\mathcal{E}$. There are fewer than $|\mathcal{E}|^{\kappa/2}$ choices of such instances and, for any such choice, each $v_i' = (a_{i,1}, \ldots, a_{i,\kappa})$ has half its entries already specified, so the remaining choices are fewer than $n^{\kappa n/2}$. This gives an upper bound of $n^{\kappa n/2} |\mathcal{E}|^{\kappa/2}$ on the number of such distributions. This number cannot be smaller than $|\mathcal{G}| \geq (n/\kappa)^{\kappa n} 2^{-s}$; therefore

$$|\mathcal{E}| \geq n^n \kappa^{-2n} 2^{-2s/\kappa}. \tag{5.1}$$

In a comparison-based decision tree model, each input instance is associated with the leaf of a binary decision tree of depth at most $2c(h+n)$, ie, with one with at most $2^{2c(h+n)}$ leaves. This would give us a lower bound on s if each instance was assigned a distinct leaf. But this may not be the case. However, we have a *collision* bound, saying that at most 4^n instances can be mapped to the same leaf. This implies that $|\mathcal{E}| 4^{-n} \leq 2^{2c(h+n)}$; and by (5.1), $s = \Omega(\kappa n \log n)$; hence the lemma.

To prove the collision bound, we use the tie-breaking rule mentioned earlier: $x_i \mapsto n x_i + i - 1$. It is clear that two instances mapping to two distinct permutations must lead to two different leaves of the decision tree. So the only question left is to bound the number of instances mapping to a given permutation. Let $x = (x_1, \ldots, x_n)$ be an input instance (no tie-breaking). Represent the ground set of this

instance as an n-bit vector α ($\alpha_i = 1$ if some $x_j = i$, else $\alpha_i = 0$). Let x be sorted to give the vector $y = (y_1, \ldots, y_n)$. For $i = 2, \ldots, n$, let $\beta_i = 1$ if $y_i = y_{i-1}$, else $\beta_i = 0$. Given the vectors α, β and the induced permutation, the input instance x can be recovered. This proves the collision bound. □

5.1.2 Learn & Sort

The V-list is built in the first $O(\log n)$ rounds. The D_i-trees will be built after $O(n^\varepsilon \log n)$ additional rounds, which will complete the training phase. During that phase, sorting is handled via, say, mergesort to guarantee $O(n \log n)$ complexity. The training part per se consists of learning basic information about H_i^V for each i. For notational simplicity, fix i and let $p_k = \text{Prob}_{\mathcal{D}_i}[\, v_k \le y < v_{k+1}\,]$. Let $M = cn^\varepsilon \log n$, for a large enough constant c. For any k, let χ_k be the number of times, over the first M rounds, that v_k is found to be the V-list predecessor of some x_i. (We use standard binary search to compute predecessors in the training phase.) Finally, define the D_i-tree to be a weighted binary search tree defined over all the v_k's such that $\chi_k > Mn^{-\varepsilon}$. Recall that the defining property of such a tree is that the node associated with a key of weight χ_k is at depth $O(\log \chi/\chi_k)$, where $\chi = \sum \chi_k$. We apply this procedure for each $i = 1, \ldots, n$.

This D_i-tree is essentially the pruned version of the one mentioned earlier. Like before, its size is $O(M/(Mn^{-\varepsilon}) = O(n^\varepsilon)$. The way we use it is similar to what we described, with a few minor differences. For completeness, we go over it again: given x_i, we perform a binary search down the D_i-tree, stopping as soon as we encounter a node whose associated key v_k is such that $x_i \in [v_k, v_{k+1})$, in which case we have the predecessor of x_i in the V-list and we are done. If we reach the bottom of the D_i-tree without success, we simply perform a standard binary search in the V-list.

Lemma 5.1.8. *Fix i. With probability at least $1 - 1/n^2$, for any k, $p_k > n^{-\varepsilon}$ implies that $Mp_k/2 < \chi_k < 3Mp_k/2$.*

Proof. The expected value of χ_k is Mp_k. If $p_k = \Omega(n^{-\varepsilon})$ then, by Chernoff's bound [11] (pages 267–268), the count χ_k deviates from its expectation by more than $a = Mp_k/2$ with probability less than

$$e^{-a^2/(2p_kM)+a^3/(2p_k^2M^2)} + e^{-a^2/(2p_kM)} < n^{-b},$$

for some constant b growing linearly with c. A union bound (over all k) completes the proof. □

Suppose the condition of the lemma holds for each k (and fixed i). We show now that the expected search time is $O(\varepsilon^{-1} H_i^V + 1)$. Consider each element in the sum $H_i^V = \sum_k p_k \log p_k^{-1}$.

- $p_k > n^{-\varepsilon}$: if v_k is in the D_i-tree, then the cost of the search is $O(\log \chi/\chi_k)$, so its contribution to the expected running time is $O(p_k \log \chi/\chi_k)$. By the lemma, this is also $O(p_k(1 + \log p_k^{-1}))$, as desired. If v_k is not in the D_i-tree,

then the search is unsuccessful and costs $O(\log n)$ time: its contribution to the expected running time is $O(p_k \log n)$. Not being in the tree, however, means that $\chi_k \leq Mn^{-\varepsilon}$; hence $p_k < 2n^{-\varepsilon}$ and the contribution is $O(\varepsilon^{-1} p_k \log p_k^{-1})$.

- $p_k \leq n^{-\varepsilon}$: the search time is always $O(\log n)$ time; hence the contribution to the expected running time is $O(\varepsilon^{-1} p_k \log p_k^{-1})$.

By summing up over all k, we find that the expected search time is $O(\varepsilon^{-1} H_i^V + 1)$. This assumes the conditions of the lemma. But these are satisfied for all i with probability at least $1 - 1/n$. This leaves a probability $1/n$ that the training fails and we are stuck with $\Theta(n \log n)$ sorting—note that we do not try to detect failure. But this adds only an additive sublinear term to the expected complexity and is therefore negligible.

5.2 Delaunay Triangulations

Let $I := (x_1, \cdots, x_n)$ denote an input instance, where each x_i is a point in the plane, generated by a point distribution $\mathcal{D}_i$. The distributions $\mathcal{D}_i$ are arbitrary, and may be continuous, although we never explicitly use such a condition. Each x_i is independent of the others, and so the input I is drawn from the product distribution $\mathcal{D} := \prod_i \mathcal{D}_i$. In each round, a new input I is drawn from $\mathcal{D}$, and we wish to compute the Delaunay triangulation of I. We are in the comparison model, so any operation consists of evaluating a polynomial at some point (more details about this are given in Section 5.3). Although it is not critical, for the sake of simplicity, we will assume that the points of I are in general position, which is true with probability one when all the $\mathcal{D}_i$'s are continuous. Also we will assume that there is a bounding box such that all points always lie inside this box.

The distribution $\mathcal{D}$ also induces a (discrete) distribution on the set of Delaunay triangulations, viewed as labelled graphs on the vertex set $[1, n]$. Consider the entropy of this distribution: for each graph G on $[1, n]$, let p_G be the probability that it represents the Delaunay triangulation of $I \in_R \mathcal{D}$. Abusing notation, let the output entropy $H(T(I)) := -\sum_G p_G \log p_G$. By information-theoretic arguments, this quantity is a lower bound on the expected time required by any comparison-based algorithm to compute the Delaunay triangulation of $I \in_R \mathcal{D}$. An *optimal* algorithm will be one that has an expected running time of $O(H(T(I)) + n)$.

Our main result is the following.

Theorem 5.2.1. *For inputs $I_1, I_2, \ldots$ drawn from the product distribution $\mathcal{D} = \prod_i \mathcal{D}_i$, and for any constant $\varepsilon > 0$, there is a self-improving algorithm for finding the Delaunay triangulations of the I_j that has a learning phase of $O(n^\varepsilon)$ rounds and uses $O(n^{1+\varepsilon})$ space [6]. The limiting running time is $O(\varepsilon^{-1}(H(T(I)) + n))$, and therefore optimal.*

[6] The total time required for the learning phase is also $O(n^{1+\varepsilon})$.

From the linear time reduction of sorting to Delaunay triangulations, the lower bounds for sorting carry over to Delaunay triangulations. As an immediate corollary of Lemma 5.1.2, we get -

Corollary 5.2.2. *There exists an input distribution $\mathcal{D}$ such that any self-improving algorithm computing the Delaunay triangulation of inputs from $\mathcal{D}$ in $O(H(D)+n)$ limiting running time requires $\Omega(2^n)$ space.*

Furthermore, by Lemma 5.1.7, the time-space tradeoff we provide is essentially optimal.

5.2.1 The algorithm

We describe the algorithm in two parts. The first part explains the learning phase and the data structures that are constructed. Then, we explain the how these data structures are used to speed up the computation in the limiting phase. As before, the expected running time will be expressed in terms of certain parameters of the data structures obtained in the learning phase. In the next section, we will prove that these parameters are comparable to the output entropy $H(\mathcal{D})$. We assume in this section that the distributions $\mathcal{D}_i$ are known to us. Furthermore, the data structures described here will use $O(n^2)$ space. Section 5.4 repeats the arguments of Section 5.1.2 to give the space-time tradeoff bounds of Theorem 5.2.1.

Learning Phase

For each round in the learning phase, we use a standard algorithm to compute the output Delaunay triangulation. We also perform some extra computation to build some data structures that will allow speedup in the limiting phase.

The learning phase is as follows. Take the first $\lambda := c\log n$ input lists $I_1, I_2, \cdots, I_k$, where c is a sufficiently large constant. Merge them into one list S of $\lambda n = cn\log n$ points. Setting $\varepsilon := 1/n$, find an ε-net V for the set of all open disks. In other words, find a set $V \subseteq S$ such that for any open disk C that contains more than $\varepsilon kn = c\log n$ points of S, C contains at least one point of V. Matousek, *et al.* show that [60] there exist ε-nets of size $O(1/\varepsilon)$ for disks , which here is $O(n)$. Furthermore, a construction and analysis similar to that of Clarkson and Varadarajan [30] yields a randomized construction (with polynomially small error probability) that takes $n(\log n)^{O(1)}$ time.

We construct the Delaunay triangulation of V, which we denote by $T(V)$. This is the equivalent of the V-list for the self-improving sorter. We build an optimal planar point location structure (called Γ) for $T(V)$: given a point, we can find in $O(\log n)$ time the triangle of $T(V)$ that it lies in. Define the random variable t_i to be the triangle of $T(V)$ that x_i falls into [7]. Now let the entropy of t_i be H_i^V. If the probability that x_i falls in triangle t of $T(V)$ is p_i^t, then $H_i^V = -\sum_t p_i^t \log p_i^t$. For each i, we construct a search structure Γ_i of size $O(n)$ that finds t_i in expected

[7] Assume that we add the vertices of the bounding box to V. This will ensure that x_i will always fall in some triangle t_i.

$O(H_i^V)$ time. These Γ_i's can be constructed using the results of Arya *et al.* [19], for which the number of primitive comparisons is $H_i^V + o(H_i^V)$). These correspond to the D_i-trees used for sorting.

We will show that the triangles of $T(V)$ do not contain many points of a new input $I \in_R \mathcal{D}$ on the average. Consider a triangle t of $T(V)$ and let C_t be its circumscribed disk; this is a Delaunay disk of V. Let $X_t := |I \cap C_t|$, the random variable that is the number of points of $I \in_R \mathcal{D}$ that fall inside C_t. Note that the randomness comes from the random distribution of S, and so V and $T(V)$, as well as the randomness of I. We are interested in the expectation $\mathbf{E}[X_t]$ over I of X_t. All expectations are taken over a random input I chosen from $\mathcal{D}$.

Claim 5.2.3. *With probability at least $1 - n^{-3}$ over the construction of $T(V)$, for every triangle t of $T(V)$, $\mathbf{E}[X_t] = O(1)$.*

Proof. This is similar to the argument given in Lemma 5.1.3 with a geometric twist. Let the list of points S be $s_1, \cdots, s_{kn}$, the concatenation of I_1 through I_k. Consider the triangle t with vertices s_1, s_2, s_3. Note that all the remaining $kn - 3$ points are chosen independently of these three, from some distribution $\mathcal{D}_j$. For each $j \in [4, kn]$, let $Y_j^{(t)}$ be the indicator variable for the event that s_j is inside C_t. Let $Y^{(t)} = \sum_j Y_j^{(t)}$. By the Chernoff bound, for any $\beta \in (0, 1]$,

$$\Pr[Y^{(t)} \leq (1 - \beta)\mathbf{E}[Y^{(t)}]] \leq e^{-\beta^2 \mathbf{E}[Y^{(t)}]/2}$$

Setting $\beta = 1/2$, if $\mathbf{E}[Y^{(t)}] > 48 \log n$, then $Y^{(t)} > 24 \log n$ with probability at least $1 - n^{-6}$. We can now consider any triangle generated by some triple of points s_i, s_j, s_m, for $i, j, m \in [4, \lambda n]$, and apply the same argument as above. Taking a union bound over all triples of the points in S, we obtain that with probability at least $1 - n^{-3}$, for any triangle t generated by the points of S, if $\mathbf{E}[Y^{(t)}] > 48 \log n$, then $Y^{(t)} > 24 \log n$. We henceforth assume that this event happens.

Consider a triangle t of $T(V)$ and its circumcircle C_t. Since $T(V)$ is Delaunay, C_t contains no point of V in its interior. Since V is a $(1/n)$-net for all disks with respect to S, C_t contains at most $c \log n$ points of S, that is, $Y^{(t)} \leq c \log n$. This implies that $\mathbf{E}[Y^{(t)}] = O(\log n)$, as in the previous paragraph. Since $\mathbf{E}[Y^{(t)}] > (\log n - 3)\mathbf{E}[X_t]$, we obtain $\mathbf{E}[X_t] = O(1)$, as claimed. □

5.2.2 Limiting Phase

We assume that we are done with learning phase, and have $T(V)$ with the property given in Claim 5.2.3: for every triangle $t \in T(V)$, $\mathbf{E}[X_t] = O(1)$. We have reached the limiting phase where the algorithm is expected to compute the Delaunay triangulation with the optimal running time. We will prove the following lemma in this section.

Lemma 5.2.4. *Using the data structures from the learning phase, and the properties of them that hold with probability $1 - O(1/n)$, in the limiting phase the Delaunay triangulation of input I can be generated in expected $O(n + \sum_{i=1}^{n} H_i^V)$ time.*

The algorithm, and the proof of this lemma, has two steps. In the first step, $T(V)$ is used to quickly compute $T(V \cup I)$, with the time bounds of the lemma. In the second step, $T(I)$ is computed from $T(V \cup I)$, using a randomized splitting algorithm proposed by Chazelle *et al* [27], whose Theorem 3 is as follows.

Theorem 5.2.5. *Given a set of n points P and its Delaunay triangulation, for any partition of P into two disjoint subsets P_1 and P_2, the Delaunay triangulations $T(P_1)$ and $T(P_2)$ can be computed in $O(n)$ expected time, using a randomized algorithm.*

The remainder of the proof of the lemma, and of this subsection, is devoted to showing that $T(V \cup I)$ can be computed in the time bound of the lemma. The algorithm is as follows. For each x_i, we use Γ_i to find the triangle t_i of $T(V)$ that contains it. By the arguments given in the previous section, this takes time $O(\sum_{i=1}^{n} H_i^V)$. We now need to argue that given the t_i's, the Delaunay triangulation $T(I)$ can be computed in expected linear time. For each x_i, we walk through $T(V)$ and find all the Delaunay disks of $T(V)$ that contain x_i, as in incremental constructions of Delaunay triangulations. This is done by breadth-first search of the dual graph of $T(V)$, starting from t_i. Let S_i denote the set of circumcircles containing x_i. The following standard claim implies that this procedure will work.

Claim 5.2.6. *The set of $t \in T(V)$ with $C_t \in S_i$ is a connected set in the dual graph of $T(V)$.*

Proof. Consider some triangle t with $C_t \in S_i$. We will show that t is connected to t_i by a path in the dual graph of $T(V)$. Consider the edge e such that x_i is in the sector bounded by C_t and e. Let t' be the neighbor of t adjacent to e. Note that since C_t is a Delaunay triangle, $t' \in S_i$. If t' is t_i, we are done. If not, then consider the edge e' such that x_i is in the sector bounded by $C_{t'}$ and e'. Refer to Figure 5.1. The edge e' is closer to x_i than e. We now consider the neighbor of t' adjacent to e' and continue in this manner. Eventually, we must reach t_i by a connected path in the dual graph of $T(V)$. □

Claim 5.2.7. *Given all t_i's, all S_i sets can be found in expected linear time.*

Proof. To find all circles contains x_i, do a breadth-first search from t_i. For any triangle t encountered, check if C_t contains x_i. If it does not, then we do not look at the neighbours of t. By Claim 5.2.6, we will visit all C_t's that contain x_i. The time taken to find S_i is $O(|S_i|)$. The total time taken to find all S_t's (once all the t_i's are found) is $O(\sum_{i=1}^{n} |S_i|)$. Define the indicator function $\chi(t, i)$ that takes value 1 if $x_i \in t$ and zero otherwise. We have

$$\sum_{i=1}^{n} |S_i| = \sum_{i=1}^{n} \sum_{t \in T(V)} \chi(t, i) = \sum_{t \in T(V)} \sum_{i=1}^{n} \chi(t, i) = \sum_{t} X_t.$$

Therefore, by Claim 5.2.3,

$$\mathbf{E}[\sum_{i=1}^{n} |S_i|] = \mathbf{E}[\sum_{t} X_t] = \sum_{t} \mathbf{E}[X_t] = O(n).$$

This implies that all S_i's can be found in expected linear time. □

Our aim is to build the Delaunay triangulation of $V \cup I$ in linear time using the S_i sets. This is done by a standard incremental construction where the x_i's are added in order $x_1, x_2, \cdots, x_n$. We will show how we can get the set of edges that each x_i will "kill" using the S_i sets. We will assume that given any triangle t, we can get all the S_i sets that t belongs to.

Let $V_i := V \cup \{x_1, \cdots, x_i\}$. When we add x_1, the edges of $T(V)$ that will be affected are the edges of triangles in S_1. Therefore, $T(V_1)$ can be obtained in $O(|S_1|)$ time. Now suppose we have $T(V_{i-1})$ and we add x_i. Again, we can show that if some edge from $T(V)$ is affected, it must be an edge of a triangle of S_i.

Claim 5.2.8. *When x_i is added to $T(V_{i-1})$, suppose that edge e is removed. Then if the endpoints of e are both in V, then e is an edge of some triangle in S_i.*

The claim above, which we shall prove shortly, shows that only $O(|S_i|)$ time is required to find edges from $T(V)$ that are removed. But now, we have the additional problem of finding affected edges which may not have an endpoint in V, and therefore are not present in $T(V)$.

Claim 5.2.9. *Suppose e is killed by x_i and has an endpoint in I. There is an edge $f \in T(V)$ such that, for the two triangles t, t' incident on f, the point x_i and the endpoints of e lie in either C_t or $C_{t'}$.*

Again, we defer this proof to the end of the section. This now gives us a method of finding edges of $T(V_{i-1})$ affected by the addition of x_i. Take a triangle $t \in S_i$ and choose an edge e of t (for ease of notation, we will say $e \in t$). Let the neighbour of t incident to e be t'. Look at the points in $\{x_1, \cdots, x_{i-1}\}$ that are in C_t and $C_{t'}$, and take the edges of $T(V_{i-1})$ between them. These are the edges that need to be checked.

Claim 5.2.10. *Given all S_i sets and $T(V)$, $T(V_n)$ can be generated in expected linear time.*

Proof. The total time taken to handle all edges of $T(V)$ that get killed is $\mathbf{E}[\sum_{i=1}^{n} |S_i|] = O(n)$. Consider some $t \in T(V)$ and edge e of t. Let $t^e \in T(V)$ be incident to e. The random variable $Z_{t,e}$ is set to be $X_t X_{t^e}$. By Claim 5.2.9, the total time to find all (other) affected edges is bounded above by

$$\sum_{i=1}^{n} \sum_{t \in S_i} \sum_{e \in t} Z_{t,e}$$

For a triangle t, we define the indicator random variable $\chi(t, i)$, as before, for the event that x_i falls in C_t. Thus, $X_t = \sum_{i=1}^{n} \chi(t, i)$.

$$\begin{aligned} \sum_{i=1}^{n} \sum_{t \in S_i} \sum_{e \in t} Z_{t,e} &= \sum_{i=1}^{n} \sum_{t} \chi(t, i) \sum_{e \in t} X_t X_{t^e} \\ &= \sum_{i=1}^{n} \sum_{t} \sum_{e \in t} \chi(t, i) X_t X_{t^e} \end{aligned}$$

$$\begin{aligned}\mathbf{E}[\chi(t,i)X_tX_{t^e}] &= \mathbf{E}\big[\chi(t,i)\sum_{j=1}^{n}\chi(t,j)\sum_{k=1}^{n}\chi(t^e,k)\big] \\ &= \sum_{j=1}^{n}\mathbf{E}[\chi(t,i)\chi(t,j)\chi(t^e,j)] + \sum_{j\neq k}\mathbf{E}[\chi(t,i)\chi(t,j)\chi(t^e,k)]\end{aligned}$$

Since $\chi(t,i)$ is an indicator, $\chi(t,j)\chi(t^e,j) \leq \chi(t,j)$. For $j \neq k$, $\chi(t,j)$ and $\chi(t^e,k)$ are independent. For the second summation in the equation above, we can separate out the case $i = j$ and $i = k$.

$$\begin{aligned}\mathbf{E}[\chi(t,i)X_tX_{t^e}] &= \sum_{j=1}^{n}\mathbf{E}[\chi(t,i)\chi(t,j)\chi(t^e,j)] + \sum_{k\neq i}\mathbf{E}[\chi(t,i)^2\chi(t^e,k)] \\ &\quad + \sum_{j\neq i}\mathbf{E}[\chi(t,i)\chi(t^e,i)\chi(t,j)] + \sum_{j\neq k\neq i}\mathbf{E}[\chi(t,i)\chi(t,j)\chi(t^e,k)] \\ &\leq \sum_{j=1}^{n}\mathbf{E}[\chi(t,i)\chi(t,j)] + \mathbf{E}[\chi(t,i)]\sum_{k\neq i}\mathbf{E}[\chi(t^e,k)] \\ &\quad + \sum_{j\neq i}\mathbf{E}[\chi(t,i)]\mathbf{E}[\chi(t,j)] + \sum_{j\neq k\neq i}\mathbf{E}[\chi(t,i)]\mathbf{E}[\chi(t,j)]\mathbf{E}[\chi(t^e,k)]\end{aligned}$$

In the first sum, we remove the term where $j = i$. In the last two terms, we remove the term dependent on i from the summation.

$$\begin{aligned}\mathbf{E}[\chi(t,i)X_tX_{t^e}] &= \mathbf{E}[\chi(t,i)] + \mathbf{E}[\chi(t,i)]\sum_{j\neq i}\mathbf{E}[\chi(t,j)] \\ &\quad + \mathbf{E}[\chi(t,i)]\sum_{k\neq i}\mathbf{E}[\chi(t^e,k)] + \mathbf{E}[\chi(t,i)]\sum_{j\neq i}\mathbf{E}[\chi(t,j)] \\ &\quad + \mathbf{E}[\chi(t,i)]\sum_{j\neq k\neq i}\mathbf{E}[\chi(t,j)]\mathbf{E}[\chi(t^e,k)]\end{aligned}$$

Finally, we upper bound the summations by adding back the terms dependent on i and then getting terms like $\mathbf{E}[X_t]$ which we have bounds for.

$$\begin{aligned}\mathbf{E}[\chi(t,i)X_tX_{t^e}] &\leq \mathbf{E}[\chi(t,i)] + 2\mathbf{E}[\chi(t,i)]\sum_{j=1}^{n}\mathbf{E}[\chi(t,j)] + \mathbf{E}[\chi(t,i)]\sum_{k=1}^{n}\mathbf{E}[\chi(t^e,k)] \\ &\quad + \mathbf{E}[\chi(t,i)]\big(\sum_{j=1}^{n}\mathbf{E}[\chi(t,j)]\big)\big(\sum_{k=1}^{n}\mathbf{E}[\chi(t^e,k)]\big) \\ &= \mathbf{E}[\chi(t,i)]\big(1 + 2\mathbf{E}[X_t] + \mathbf{E}[X_{t^e}] + \mathbf{E}[X_t]\mathbf{E}[X_{t^e}]\big)\end{aligned}$$

By Claim 5.2.3, we get that $\mathbf{E}[\chi(t,i)X_tX_{t^e}] \leq \alpha\mathbf{E}[\chi(t,i)]$, for some fixed constant

α. The expected running time is bounded by

$$\begin{aligned}
\mathbf{E}[\sum_{i=1}^{n}\sum_{t\in S_i}\sum_{e\in t} Z_{t,e}] &= \sum_{i=1}^{n}\sum_{t}\sum_{e\in t}\mathbf{E}[\chi(t,i)X_t X_{t^e}] \\
&\leq \alpha\sum_{i=1}^{n}\sum_{t}\sum_{e\in t}\mathbf{E}[\chi(t,i)] \\
&= 3\alpha\sum_{i=1}^{n}\mathbf{E}[\sum_{t}\chi(t,i)] \\
&= 3\alpha\sum_{i=1}^{n}\mathbf{E}[|S_i|] = O(n)
\end{aligned}$$

□

With this claim, it follows that $T(V_n)$ can be computed in expected $O(n + \sum_{i=1}^{n} H_i^V)$ time, and hence, as discussed at the beginning of this subsection, Lemma 5.2.4 follows. Using the same ideas, we prove another claim which will be useful in later proofs.

Claim 5.2.11. *For all $j \leq i$, the expected degree of x_j in $T(V_i)$ is $O(\mathbf{E}[|S_j|])$.*

Proof. In the incremental construction of $T(V_n)$, the degree of x_j never decreases. Therefore, it suffices to bound the degree of x_j in $T(V_n)$. Let us perform the incremental construction such that such that x_j is the last vertex to be added. The edges adjacent to x_j involving vertices of V are atmost $|S_j|$. Let us now bound edges connecting x_j to other vertices of I. This can certainly be bounded (upto a multiplicative constant factor) by the number of edges that x_j kills. By the arguments given earlier, this is -

$$\sum_{t\in S_j}\sum_{e\in t} Z_{t,e'} = O(|S_j|)$$

□

We end by giving proofs for Claims 5.2.8 and 5.2.9.

Proof. (Claim 5.2.8) The edge e must be an edge in $T(V)$. Also, e is an edge of triangle t in $T(V_{i-1})$ and is not a boundary edge. Refer to Figure 5.1. The point x_i is in the sector bounded by C_t and e. Since $e \in T(V)$, there must be a point $y \in V$ such that e and y form a triangle t' of $T(V)$ and x_i and y are on the same side of e. The point y cannot be inside C_t, since t is a Delaunay triangle of $T(V_{i-1})$. Therefore, the angle subtended by y at e is smaller than that of x_i. The circle $C_{t'}$ must contain x_i and $t' \in S_i$. □

Proof. (Claim 5.2.9) Suppose some edge e in triangle $t \in T(V_{i-1})$ is killed by x_i. We will denote the vertices of t by u_1, u_2, and u_3. The point x_i is inside C_t. Consider the set of edges of $T(V)$ that intersect C_t. We can impose a natural linear ordering

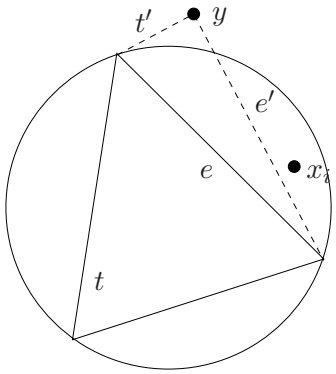

Figure 5.1: Proofs of Claims 5.2.6 and 5.2.8

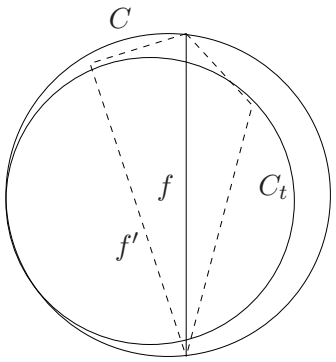

Figure 5.2: Proof of Claim 5.2.9

on these edges. If none of these edges separate the set points $U = \{u_1, u_2, u_3, x_i\}$, then they all lie in a triangle t of $T(V)$ and we are done.

Consider the last edge (in the order from left to right) f that separates U into U_ℓ and U_r. Refer to Figure 5.2. The next edge after f to the right (in the ordering) that intersects C_t does not separate U. Let the two triangles of $T(V)$ that share f as an edge be t_ℓ, t_r (the left and right triangles). The triangle t_r contains U_r. Let C be the circle tangential to C_t at the left of f and having f as a chord. (Note that if f is actually an edge of t, then C_t and C are just the same, and we will end up with Claim 5.2.8.) If the vertex of t_ℓ not on f lies outside C, then C_{t_ℓ} will contain all of C_t which lies to the left of f. This implies that C_{t_ℓ} contains U_ℓ. The situation is as follows: triangles t_ℓ and t_r in $T(V)$ share edge f. Edge f divides U into U_ℓ and U_r and they are contained in C_{t_ℓ} and C_{t_r}, respectively, proving the claim (for this case).

Suppose the third vertex of t_ℓ is inside C. The triangle t_ℓ is shown by the dashed triangle in Figure 5.2 to the left of f. Let the angle to f be θ_ℓ. Let the angle subtended by any point in the right part of C_t be θ_r. Note that $\theta_\ell + \theta_r > \pi$.

Therefore, the circle C_{t_ℓ} will contain the right part of C_t (and, as a result, U_r).

The edge f' is in $T(V)$ and intersects C_t. Also, f' is larger than f in the ordering of edges. If f' does not separate U, then C_{t_ℓ} must contain U_ℓ and we are done. If not, then suppose f' divides U into U'_ℓ and U'_r. The triangle t_ℓ is actually to the right of f' and C_{t_ℓ} contains U'_r. This leaves us in a situation analagous to f: the circumcircle of triangle to the right of f' contains all points in U to the right of f'. Therefore, we can apply the same argument as above: either we will stop, getting our desired triangles, or we will move to the edge to the right of f'. □

5.3 Running time analysis

In this section, we prove the running time bound in Lemma 5.2.4 is indeed optimal. Before we get into the analysis of the various entropies that represent the running time, it is important to clarify the model of computation. We are using comparison based algorithms, where a single step (or "comparison") involves evaluating a point $(z_1, z_2, \cdots, z_d) \in \mathbb{R}^d$ (for constant d) at some polynomial $f(z_1, z_2, \cdots, z_d) : \mathbb{R}^d \to \mathbb{R}$ and checking if the result is positive or negative. Based on this result, the algorithm chooses the next comparison to make. An algorithm can be completely represented by a decision tree, with each node representing some comparison. In this model, we get an information-theoretic lower bound of $H(T(I))$ for computing the Delaunay triangulation of input $I \in_R \mathcal{D}$.

Recall that by Lemma 5.2.4, the running time of the our algorithm is expected $O(n + \sum_i H_i^V)$. The aim of this section is to prove the optimality of the algorithm by the following theorem.

Theorem 5.3.1. *For H_i^V, the entropy of the triangle t_i of $T(V)$ containing x_i, and $H(T(I))$, the entropy of the Delaunay triangulation of I, considered as a labelled graph,*

$$\sum_i H_i^V = O(H(T(I)) + n).$$

This theorem will be proven through a chain of lemmas, which will eventually connect $\sum_{i=1}^n H_i^V$ to $H(T(I))$. Note that V is a fixed set and there is no randomness in $T(V)$. As a result, for the sake of information theory bounds, we can assume that $T(V)$ is known in advance: indeed, any computation whatsoever can be done in advance on the points in V and is not charged as a comparison.

The chain of lemmas begins with $H(T(V_n))$, which is bounded above by $O(H(T(I)+n)$ in the lemma below. The entropy $H(T(V_n))$ is used to bound $\sum_i \mathbf{E}[H_i]$ in the next lemma, where H_i is the entropy of w_i, the triangle of $T(V_{i-1})$ that contains x_i. After some preliminary lemmas, the final lemma in the chain uses $\sum_i \mathbf{E}[H_i]$ to bound $\sum_i H_i^V$, as needed for the theorem.

By analogy to $H(T(I))$, let $H(T(V_n))$ be the entropy of $T(V_n)$ as a labelled graph, under the distribution induced by that of I. (Recall that $V_n := V \cup I$.) The entropy $H(T(V_n))$ is a lower bound for the expected running time of any comparison-based algorithm that computes $T(V_n)$.

The first lemma in the chain is the following.

Lemma 5.3.2.

$$H(T(V_n)) = O(H(T(I)) + n).$$

Proof. Using Chazelle's linear-time algorithm to compute the intersection of two three-dimensional convex polyhedra [26], we can compute $T(V_n)$ in $O(n)$ time, given $T(V)$ and $T(I)$. Suppose we represent every graph induced by a Delaunay triangulation on n points by some string, denoted by $s(T)$. By information theory, there exists some string encoding such that $\mathbf{E}[|s(T(I))|] = O(H(T(I)))$. Suppose, for input I, we are given the string $s(T(I))$, so we can uniquely identify $T(I)$. Now, we use the linear-time algorithm to compute $T(V_n)$. Obviously, this algorithm only perform $O(n)$ comparisons. Therefore, the output $T(V \cup I)$ can be uniquely identified by $s(T(I))$ and cn more bits, for some constant c. We have $\mathbf{E}[|s(T(I))| + cn] \geq H(T(V_n))$. This completes the proof. □

Let us consider an incremental construction of $T(V_n)$. At the ith step, x_i is added to $T(V_{i-1})$. We can consider a random process associated with this step. The points $x_1, \cdots, x_{i-1}$ are already fixed, thereby fixing $T(V_{i-1})$. We can consider the entropy of the random variable w_i that is the triangle of $T(V_{i-1})$ in which x_i falls. More precisely, we define

$$H_i^{T(V_{i-1})} := H(w_i) = -\sum_{t \in T(V_{i-1})} p(t,i) \log p(t,i)$$

where $p(t,i)$ is the probability that x_i lies in t. Note that this entropy itself is a random variable, since $T(V_{i-1})$ depends on $x_1, \cdots, x_{i-1}$ which are randomly chosen. But w_i is independent of this randomness (since the distributions $\mathcal{D}_i$'s are all independent). Therefore, we can take the expectation over the random choices $\{x_1, \cdots, x_{i-1}\}$ $\mathbf{E}_{x_1,\cdots,x_{i-1}}[H_i^{T(V_{i-1})}]$. Again, let us explain what this means. Given any set of points $x_1, \cdots, x_{i-1}$, we can define the entropy $H_i^{T(V_{i-1})}$. Now, because the randomness of w_i only depends on the randomness of x_i, w_i is independent of $x_1, \cdots, x_{i-1}$. Obviously, $H_i^{T(V_{i-1})}$ is a function of $x_1, \cdots, x_{i-1}$. We take the expectation over the random choices of $x_1, \cdots, x_{i-1}$ to get $\mathbf{E}[H_i^{T(V_{i-1})}]$. For clarity, we drop the subscripts and denote this by $\mathbf{E}[H_i]$.

In the next lemma, we relate the entropy of this incremental procedure of constructing $T(V_n)$ to the actual entropy of the $T(V_n)$.

Lemma 5.3.3.

$$\sum_i \mathbf{E}[H_i] = O(H(T(V_n)) + n)$$

To prove this, we use Claim 5.2.11 (restated for convenience) and a lemma about joint entropies.

Claim 5.3.4. *For all $j \leq i$, the expected degree of x_j in $T(V_i)$ is $O(\mathbf{E}[|S_j|])$.*

Lemma 5.3.5.

$$H(w_1, \cdots, w_n) \geq \sum_{i=1}^{n} \mathbf{E}[H_i]$$

Here the $H(w_1, \cdots, w_n)$ is the joint entropy of all $w_1, \cdots, w_n$, and a lower bound on the expected length of any string representation of $w_1, \cdots, w_n$.

Proof. We will prove by induction on k that $H(w_1, \cdots, w_k) \geq \sum_{i=1}^{k} \mathbf{E}[H_i]$. Recall that w_k is the triangle of $T(V_{k-1})$ that contains x_k. As in Claim 5.1.5, this is a consequence of the independence of the x_i's. The reason why we cannot immediately use independence is that the domain random variable w_i *does* depend on the choices of $x_1, \cdots, x_{k-1}$, because w_k is a triangle of $T(V_{k-1})$. In some sense, we are just stating a well known fact about conditional entropies, but this small technical problem forces us to reprove it for our setting. We proceed with a proof by induction.

base case: For $k = 1$, $H(w_1) = H_1$ (note that H_1 is not a random variable).

induction step: Assume the claim is true up to $k-1$. For any triangle t (which is specified by a triple of vertex labels), let $p(i, t)$ be the probability that x_i falls in t. Suppose that $x_1, \cdots, x_{k-1}$ are fixed. We have

$$H_k = -\sum_{t \in T(V_{k-1})} p(t, k) \log p(t, k).$$

Let $\gamma(t, k-1)$ be the indicator variable of the event that $T(V_{k-1})$ has triangle t. This is a random variable, depending on $x_1, \cdots, x_{k-1}$. Removing the assumption that $x_1, \cdots, x_{k-1}$ are fixed, we take the expectation of H_k:

$$\begin{aligned} \mathbf{E}[H_k] &= -\mathbf{E}[\sum_t \gamma(t, k-1) p(t, k) \log p(t, k)] \\ &= -\sum_t \mathbf{E}[\gamma(t, k-1)] p(t, k) \log p(t, k) \end{aligned}$$

Consider some sequence of triangles $\Delta_k = \langle t_1, \cdots, t_k \rangle$. For $i \leq k$, let $\mathcal{E}_i(\Delta_i)$ denote the event that $w_1 = t_1, w_2 = t_2, \cdots, w_i = t_i$.

$$\begin{aligned} \mathbf{Pr}[\mathcal{E}_k(\Delta_k)] &= \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \times \mathbf{Pr}[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\ &= \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \times p(t_k, k) \mathbf{Pr}[\gamma(k-1, t_k) = 1 \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \end{aligned}$$

This is a consequence of the independence of x_k from $x_1, \cdots, x_{k-1}$. As a result, the probability $p(t_k, k)$ is not affected by the values of $w_1, \cdots, w_{k-1}$. Note also that $\mathbf{Pr}[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_k)] = \mathbf{Pr}[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})]$. Taking the convention that

$0 \log 0 = 0$, we express the entropy $H(w_1, \cdots, w_k)$ as a sum.

$$
\begin{aligned}
& H(w_1, \cdots, w_k) \\
&= -\sum_{\Delta_k} \mathbf{Pr}[\mathcal{E}_k(\Delta_k)] \log \mathbf{Pr}[\mathcal{E}_k(\Delta_k)] \\
&= -\sum_{\Delta_k} \mathbf{Pr}[\mathcal{E}_k(\Delta_k)] \log (\mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \times p(t_k, k) \mathbf{Pr}[\gamma(k-1, t_k) = 1 \big| \mathcal{E}_{k-1}(\Delta_{k-1})]) \\
&= -\sum_{\Delta_k} \mathbf{Pr}[\mathcal{E}_k(\Delta_k)] \big(\log (\mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})]) \\
&\quad + \log p(t_k, k) + \log (\mathbf{Pr}[\gamma(t_k, k-1) = 1 \big| \mathcal{E}_{k-1}(\Delta_{k-1})]) \big)
\end{aligned}
$$

We now open the parentheses and consider each sum separately.

$$
\begin{aligned}
& -\sum_{\Delta_k} \mathbf{Pr}[\mathcal{E}_k(\Delta_k)] \log (\mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})]) \\
&= -\sum_{\Delta_k} \log (\mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})]) \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \times \mathbf{Pr}[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{\Delta_{k-1}, t_k} \log (\mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})]) \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \times \mathbf{Pr}[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{\Delta_{k-1}} \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \log (\mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})]) \sum_{t_k} \mathbf{Pr}[w_k = t_k \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{\Delta_{k-1}} \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \log (\mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})]) \times 1 \\
&= H(w_1, \cdots, w_{k-1}).
\end{aligned}
$$

Now consider the next sum in a similar manner:

$$
\begin{aligned}
& -\sum_{\Delta_k} \log p(t_k, k) \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \times p(t_k, k) \mathbf{Pr}[\gamma(t_k, k-1) = 1 \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{t_k} p(t_k, k) \log p(t_k, k) \sum_{\Delta_{k-1}} \mathbf{Pr}[\mathcal{E}_{k-1}(\Delta_{k-1})] \ \mathbf{Pr}[\gamma(t_k, k-1) = 1 \big| \mathcal{E}_{k-1}(\Delta_{k-1})] \\
&= -\sum_{t_k} p(t_k, k) \log p(t_k, k) \sum_{\Delta_{k-1}} \mathbf{Pr}[\gamma(t_k, k-1) = 1] \\
&= -\sum_{t_k} \mathbf{E}[\gamma(t_k, k-1)] p(t_k, k) \log p(t_k, k) = \mathbf{E}[H_k]
\end{aligned}
$$

The third sum is always non-negative [8]. This implies that

$$
H(w_1, \cdots, w_k) \geq H(w_1, \cdots, w_{k-1}) + \mathbf{E}[H_k] \geq \sum_{i=1}^{k} \mathbf{E}[H_i].
$$

□

[8] Indeed, we can show that this term is always zero, thereby proving equality.

Proof. (of Lemma 5.3.3) Before giving the details of the proof, let us first sketch out the main idea. Suppose all the random choices $x_1, \cdots, x_n$ have been made. We would like to argue that if we know $T(V_n)$, then in linear time we can determine the w_i's for all i. This will be done by a procedure that goes backwards: it first removes x_n, and then computes the Delaunay triangulation $T(V_{n-1})$. This can be done in time linear in the degree of x_n [4]. The triangle w_n can be determined in time linear in the degree of x_n. Now, we remove x_{n-1} and so on, thereby finding all w_i's. It seems that by a standard backwards analysis argument, we should remove the x_i's in random order. By a planarity argument, we should get that the expected degree (over the random order) is constant at every step. But because we remove only the points in I, which is a strict *subset* of V_n, this argument will not hold.

Using the properties of V and the randomness of I, we can argue that these degrees will be expected constant. From Claim 5.2.11, it is easy to see that we can get the w_i's in $O(\sum_i \mathbf{E}[|S_i|])$ time. Let us now apply an argument similar to that in Lemma 5.3.2. Let there be a string representation $s(T(V_n))$ for each possible $T(V_n)$. By definition of entropy, we can assume that $\mathbf{E}[|s(T(V_n))|] = O(H(T(V_n)))$. Using the procedure described above, we can uniquely identify $w_1, \cdots, w_n$ by a string of expected length $\mathbf{E}[|s(T(V_n))|] + O(\mathbf{E}[\sum_i |S_i|]) = O(H(T(V_n)) + n)$.

The proof now follows by Lemma 5.3.5, since $H(w_1, \cdots, w_n)$ is no more than $\mathbf{E}[|s(T(V_n))|$. □

We now come to the final lemma in our chain of entropy inequalities.

Lemma 5.3.6.

$$\sum_i H_i^V = O(\sum_i \mathbf{E}[H_i] + n)$$

Proof. Consider $x_1, \cdots, x_{i-1}$ to be chosen, fixing the triangulation $T(V_{i-1})$. The entropy H_i is now well defined. As before, $w_i \in T(V_{i-1})$ and $t_i \in T(V)$ are the triangles that x_i falls into. We will describe a procedure that given w_i finds t_i using $O(|S_i|)$ comparisons. First, we look at the Delaunay triangulations as 3-dimensional polytopes. By projecting onto the paraboloid $z = x^2+y^2$, each point of the Delaunay triangulation is represented by a halfspace in 3-dimensions. Every vertex of the polytope corresponds to a Delaunay triangle (or disk). Abusing notation, $T(V)$ and $T(V_{i-1})$ are going to be the respective polytopes. We start by tetrahedralizing $T(V)$. Since $T(V_{i-1})$ is completely contained in $T(V)$, for every vertex of $T(V_{i-1})$, we can determine a tetrahedron of $T(V)$ that contains it (maybe on the boundary). Since we are dealing with H_i^V and $\mathbf{E}[H_i]$, we do not consider comparisons that deal with vertices other than x_i. All of this can be done *before* we look at point x_i and are therefore not counted. Given w_i, we can determine the tetrahedron that it lies in without any comparisons. Since the triangle w_i will certainly be destroyed on the addition of x_i, the vertex corresponding to w_i (in the polytope) will be removed by the addition of the plane x_i. Obviously, there is some vertex of the tetrahedron (that corresponds to w_i) would also be removed by the addition of x_i to $T(V)$. In a constant number of queries, this vertex can be determined. Now, let us go back to the Delaunay triangulations. This vertex corresponds to some Delaunay disk of

$T(V)$ killed by x_i. By doing a walk through $T(V)$, we can find t_i in $O(|S_i|)$ time. This implies that

$$H_i^V \leq H_i + O(|S_i| + 1).$$

Taking expectations over I and summing,

$$\sum_i H_i^V \leq \sum_i \mathbf{E}[H_i] + O(\mathbf{E}[\sum_i |S_i|] + n) \leq \sum_i \mathbf{E}[H_i] + O(n).$$

□

As discussed above, Theorem 5.3.1 now follows by combining Lemmas 5.3.2, 5.3.3, and 5.3.6.

5.4 The time-space tradeoff

We show how to remove the assumption that we have prior knowledge of the $\mathcal{D}_i$'s (to build the search trees Γ_i) and prove the time-space tradeoff given in Theorem 5.2.1. These techniques are identical to those used in Section 5.1.2. For the sake of clarity, we give a detailed explanation for this setting. Let $\varepsilon > 0$ be any constant. The first $O(\log n)$ rounds of the learning phase are used as before to construct the Delaunay triangulation $T(V)$. We first build a standard search structure Γ over the triangles of $T(V)$. Given a point x, we can find the triangle of $T(V)$ that contains x in $O(\log n)$ time.

The learning phase goes on for $O(n^\varepsilon \log n)$ rounds. The main trick is to observe that (up to constant factors), the only probabilities that are relevant are those that are $> n^{-\varepsilon}$. In each round, for each x_i, we record the triangle of $T(V)$ that x_i falls into. At the end of $O(n^\varepsilon \log n)$ rounds, we take the set R_i of triangles such that for $t \in R_i$, x_i was in t for at least $\Omega(\log n)$ rounds. We remind the reader that $p(t, i)$ is the probability that x_i lies in triangle t. For every triangle in R_i, we have an estimate of the probability $\hat{p}(t, i)$ (obtained by simply taking the total number of times that x_i lay in t, divided by the total number of rounds). By a standard Chernoff bound argument, for all $t \in R_i$, $\hat{p}(t, i) = \Theta(p(t, i))$. Furthermore, for any triangle t, if $p(t, i) = \Omega(n^{-\varepsilon})$, then $t \in R_i$.

For each x_i, we build the approximate search structure Γ_i. Consider the following probability distribution $\bar{p}_i$ over the triangles of $T(V)$: if $t \in R_i$, set $\bar{p}(t, i) := \hat{p}(t, i)/N_i$, where $N_i := \sum_{t \in R_i} \hat{p}(t, i)$, and otherwise $\bar{p}(t, i) := 0$. Using the construction of [19], we can build the optimal planar point location structure Γ_i according to the distribution $\bar{p}_i$. The limiting phase uses these structures to find t_i for every x_i: given x_i, we use Γ_i to search for it. If the search does not terminate in $\log n$ steps or Γ_i fails to find t_i (since $t_i \notin R_i$), then we use the standard search structure, Γ, to find t_i. Therefore, we are guaranteed to find t_i in $O(\log n)$ time. Without loss of generality, we can assume that each Γ_i deals with only n^ε triangles (and therefore, a planar subdivision of size n^ε). By the bounds given in [19], each Γ_i can be constructed with size n^ε in $n^\varepsilon \log n$ time. The total space is bounded by $n^{1+\varepsilon}$ and the time required to build them is at most $n^{1+\varepsilon} \log n$.

Now we just repeat the argument given in Section 5.1.2. Instead of doing it through words, we write down the expressions (for some variety). Let $s(t,i)$ denote the time to search for x_i given that $x_i \in t$. By the properties of Γ_i, and noting that $N_i \leq 1$,

$$\begin{aligned}
\sum_{t \in R_i} \bar{p}(t,i) s(t,i) &= \sum_{t \in R_i} \bar{p}(t,i) \log(1/\bar{p}(t,i)) \\
&= N_i^{-1} \sum_{t \in R_i} \hat{p}(t,i) \log(N_i/\hat{p}(t,i)) \\
&= N_i^{-1} \big[\sum_{t \in R_i} \hat{p}(t,i) \log N_i - \sum_{t \in R_i} \hat{p}(t,i) \log \hat{p}(t,i) \big] \\
&\leq -N_i^{-1} \sum_{t \in R_i} \hat{p}(t,i) \log \hat{p}(t,i) \\
&= O\big(N_i^{-1} (- \sum_{t \in R_i} p(t,i) \log p(t,i) + 1) \big)
\end{aligned}$$

We now bound the expected search time for x_i.

$$\begin{aligned}
\sum_t p(t,i) s(t,i) &= \sum_{t \in R_i} p(t,i) s(t,i) + \sum_{t \notin R_i} p(t,i) s(t,i) \\
&= O(\sum_{t \in R_i} \hat{p}(t,i) s(t,i) + \sum_{t \notin R_i} p(t,i) \log n) \\
&= O\big(N_i \sum_{t \in R_i} \bar{p}(t,i) s(t,i) + \sum_{t \notin R_i} p(t,i) \log n \big)
\end{aligned}$$

Noting that for $t \notin R_i$, $p(t,i) = O(n^{-\varepsilon})$ and therefore $\log p(t,i) \leq -\varepsilon \log n + O(1)$, and so

$$\begin{aligned}
\sum_t p(t,i) s(t,i) &= O\big((- \sum_{t \in R_i} p(t,i) \log p(t,i) + 1) + \sum_{t \notin R_i} p(t,i) \varepsilon^{-1} (- \log p(t,i) + 1) \big) \\
&= O\big(\varepsilon^{-1} (- \sum_t p(t,i) \log p(t,i) + 1) \big) \\
&= O\big(\varepsilon^{-1} (H_i^V + 1) \big)
\end{aligned}$$

The total expected search time is $O(\varepsilon^{-1}(\sum_i H_i^V + n))$. By the analysis of Section 5.2.1 and Theorem 5.3.1, we have that the expected running time in the limiting phase is $O(\varepsilon^{-1}(H(\mathcal{D}) + n))$. This completes the proof of Theorem 5.2.1.

Chapter 6

Further directions

In this thesis, we proprosed the models of distributed property reconstruction and self-improving algorithms, and investigated certains instances of problems in these models. It is natural that the reconstruction results should raise the question of what other properties can be reconstructed. Similarly, we can ask about other problems that can be studied in the self-improving model. In this chapter, we shall discuss some specific questions that are more immediately inspired by our results. Some of these questions are of independent interest, and do not pertain to our models.

6.1 Monotonicity reconstruction

We studied the monotonicity reconstruction of functions of the form $f : [1, n]^d \to \mathbb{R}$, and designed a data structure of intervals useful for sublinear time computation. It appears that this might be useful for sublinear time estimation algorithms, say, to estimate the distance to monotonicity. Such algorithms are known [5, 68], but it is possible that some of our ideas can get better approximation factors.

Our filter's running time and error blow-up is exponential in d. Is there some way we can get any of these to be polynomial in d or is there some lower bound? It seems that completely different techniques would be required. Related to this is monotonicity reconstruction on the boolean hypercube : the functions are of the form $f : \{0, 1\}^d \to \mathbb{R}$. Can we get distributed filters (whose time is sublinear in the domain size 2^d, not necessarily polynomial in d)? A more ambitious aim would be to attempt to design a filter for monotonicity on any partial order[1]. We are allowed to preprocess the partial order (given as a DAG) in polynomial time to build any necessary data structures - this would allow us, for instance, to explicitly get all intervals - but the function would be presented in the same oracle fashion. It would be interesting if we can get some error blow-up that depends solely on some parameter of the partial order.

[1]Think of the partial order as represented by a DAG. We would probably want to allow running times that are polynomial in degrees of the elements.

6.2 Convexity reconstruction

By far the most intriguing question to come out of this work deals with planar separators. We are given a planar graph (with n vertices) as a straight line embedding represented as a DCEL. In other words, we have access to the real coordinates of the vertices, have the edges incident to a vertex in circular order, and can traverse the boundary edges of any face. Can we find a $O(\sqrt{n})$-sized vertex separator in $o(n)$ time? We showed that this is possible if the enbedding has constant aspect ratio. Not all planar graphs even have such an embedding. This question seems to be very difficult and will require radically different ideas. It seems rather unlikely that the standard toolkit of computational geometric methods will help us.

6.3 Expander reconstruction

One of the main direction of future research is improvement of the conductance bound of G' to $\Omega(\phi^2)$ (instead of $\Omega(\phi^2/\log n)$). For this, we need to have definitions of weak/strong that distinguish vertices on the basis of very small probability differences (much smaller than we currently do). These differences can be algorithmically tested, but to ensure that not too many edges are added to get G', we would need much stronger results about walks in noisy expanders.

It seems highly likely that the algorithmic procedures we use here could be used for efficient graph partitioning algorithms [13–15]. The partitions would be decided by performing random walks from vertices, and might lead to easier proofs of earlier results. We may also be able to generate sublinear routines which can implicitly represent such a partition: answering queries such as whether two vertices are in the same graph of the partition or not. Improved conductance bounds for reconstruction may lead to better graph partitioning algorithms.

6.4 Self-improving algorithms

A natural problem to attack is that of a self-improving algorithm for computing the convex hull of n points in the Euclidean plane. Again, let us assume that each input point is generated independently. Even with this restriction, we can get a large variety of distributions (with a wide range of entropies). As the other problems that we have seen, there is an information-theoretic lower bound of $\Omega(n \log n)$, and our lower bounding and information theoretic techniques (for self-improving algorithms) would certainly work here. What makes this problem different - and difficult - is the issue of *output sensitivity*. The output of a convex hull algorithm is a ordered list of the points on the hull and for every other point, a certificate that it is not on the hull. Different parts of the input are treated in a different way, and this can create many dependencies between the points. This complicates the construction of a self-improving algorithm, as we would have to exploit these dependencies to get an optimal algorithm.

A more interesting problem would be dealing with Voronoi diagrams in 3D. Here, the size of the output can be anything from linear to quadratic. There are low entropy distributions that generate low complexity outputs, and it would be very interesting to design a self-improving algorithm that can exploit that. Such an algorithm may even have some practical impact. This would require a major extension to our present set of techniques, but it is possible that our current overall approach (an "average" output constructed by the learning phase, using which every output is generated incrementally) will work.

To move away from the independent distributions we use, one could also consider time-varying distributions or Markov models. Of course, a purely adversarial model might easily defeat self-improvement: it would observe how the improvement proceeds and render it ineffective by tailoring distributions changing over time.

Bibliography

[1] P. K. Agarwal and P. K. Desikan. An efficient algorithm for terrain simplification. In *Proceedings of the 8th Annual Symposium on Discrete Algorithms (SODA)*, pages 139–147, 1997.

[2] P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. In *Proceedings of the 10th Annual Europeam Symposium on Algorithms (ESA)*, pages 29–41, 2002.

[3] P. K. Agarwal and S. Suri. Surface approximation and geometric partitions. *SIAM Journal of Computing*, 27:1016–1035, 1998.

[4] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor. A linear time algorithm for computing the voronoi diagram of a convex polygon. *Discrete and Computational Geometry*, 4:591–604, 1989.

[5] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. In *Proceedings of the 8th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 229–236, 2004.

[6] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property preserving data reconstruction. In *Proceedings of the 15th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 16–27, 2004.

[7] S. Albers and J. Westbrook. Self organizing data strutures. *Online Algorithms: The State of the Art*, 1442:13–41, 1998.

[8] D. J. Aldous and J. Fill. *Time-reversible Markov chains and random walks on graphs*. (book in preparation).

[9] B. Allen and I. Munro. Self-organizing bimary search trees. *Journal of the ACM*, 25:526–535, 1978.

[10] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 240–250, 2000.

[11] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley, 2nd edition, 2000.

[12] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the 16th Annual Symposium on Computational Geometry (SoCG)*, pages 213–222, 2000.

[13] R. Andersen. A local algorithm for finding dense subgraphs. In *Proceedings of the 19th Annual Symposium of Discrete Algorithms (SODA)*, pages 1003–1009, 2008.

[14] R. Andersen, F. R. K. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Proceedings of the Annual 47th Foundations of Computer Science (FOCS)*, pages 475–486, 2006.

[15] R. Andersen and K. Lang. An algorithm for improving graph partitions. In *Proceedings of the 19th Annual Symposium of Discrete Algorithms (SODA)*, pages 651–660, 2008.

[16] S. Arora and M. Sudan. Improved low-degree testing and its applications. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 485–495, 1997.

[17] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of np. *Journal of the ACM*, 45(1):70–122, 1998.

[18] Sanjeev Arora and Shmuel Safra. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[19] S. Arya, T. Malamatos, D. M. Mount, and K.-C. Wong. Optimal expected-case planar point location. *SIAM Journal of Computing*, 37:584–610, 2007.

[20] T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. Testing that distributions are close. In *Proceedings of the 41st Annual Foundations of Computer Science (FOCS)*, pages 259–269, 2000.

[21] I. Benjamini, O. Schramm, and A. Shapira. Every minor-closed property of sparse graphs is testable. Technical Report arXiv:0801.2797v2, arxiv, 2008.

[22] J. R. Bitner. Heuristics that dynamically organize data structures. *SIAM Journal of Computing*, 8:82–110, 1979.

[23] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[24] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[25] T. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*, pages 1196–1202, 2006.

[26] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal of Computing*, 21:671–696, 1992.

[27] B. Chazelle, O. Devillers, F. Hurtado, M. Mora, V. Sacristan, and M. Teillaud. Splitting a delaunay triangulation in linear time. *Algorithmica*, 34:39–46, 2002.

[28] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. In *Proceedings of 35th Annual Symposium on Theory of Computing (STOC)*, pages 531–540, 2003.

[29] V. Chvatal and G. Klincsek. Finding largest convex subsets. *Congressus Numerantium*, 29:453–460, 1980.

[30] K. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete and Computation Geometry*, 37:43–58, 2007.

[31] A. Czumaj, F. Ergun, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Sublinear-time approximation of euclidean minimum spanning tree. In *Proceedings of 14th Annual Symposium on Discrete Algorithms (SODA)*, pages 813–822, 2003.

[32] A. Czumaj and C. Sohler. Property testing with geometric queries. In *Proceedings of 9th Annual European Symposium on Algorithms (ESA)*, pages 266–277, 2001.

[33] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of 36th Annual Symposium on Theory of Computing (STOC)*, pages 175–183, 2004.

[34] A. Czumaj and C. Sohler. On testable properties in bounded degree graphs. In *Proceedings of the 18th Annual Symposium on Discrete Algorithms (SODA)*, pages 494–501, 2007.

[35] A. Czumaj and C. Sohler. Testing expansion in bounded degree graphs. In *Proceedings of the Annual 48th Symposium on Foundations of Computer Science (FOCS)*, pages 570–578, 2007.

[36] A. Czumaj, C. Sohler, and M. Ziegler. Property testing in computational geometry. In *Proceedings of 8th Annual European Symposium on Algorithms (ESA)*, pages 155–166, 2000.

[37] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry : Algorithms and Applications*. Springer, 2nd edition, 1998.

[38] R. Diestel.

[39] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, and D. Ron. Improved testing algorithms for monotonicity. In *Proceedings of the 3rd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 97–108, 1999.

[40] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, 2nd edition, 2000.

[41] F. Ergun, S. Kannan, S. Ravi Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. In *Proceedings of 30th Annual Symposium on Theory of Computing (STOC)*, pages 259–268, 1998.

[42] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24:441–476, 1992.

[43] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of EATCS*, 75:97–126, 2001.

[44] M. L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1:355–361, 1976.

[45] O. Goldreich. Combinatorial property testing - a survey. *Randomization Methods in Algorithm Design*, pages 45–60, 1998.

[46] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20:301–337, 2000.

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

[48] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[49] O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. Technical report, ECCC, 2000.

[50] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[51] G.H. Gonnet, J.I. Munro, and H. Suwanda. Exegesis of self-organizing linear search. *SIAM Journal of Computing*, 10:613–687, 1981.

[52] J.H. Hester and D.S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17:295–311, 1985.

[53] P. Indyk. Sublinear-time algorithms for metric space problems. In *Proceedings of 31st Annual Symposium on Theory of Computing (STOC)*, pages 428–434, 1999.

[54] P. Indyk. A sublinear-time approximation scheme for clustering in metric spaces. In *Proceedings of 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 154–159, 1999.

[55] S. Kale and C. Seshadhri. Testing expansion in bounded degree graphs. In *To appear in ICALP*, 2008.

[56] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[57] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[58] L. Lovász and M. Simonovits. The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In *Proceedings of the 31st Annual Foundations of Computer Science (FOCS)*, pages 346–354, 1990.

[59] L. Lovász and P. Winkler. Mixing times. *Microsurveys in Discrete Probability, DIMACS Series in Discrete Math and Theoretical Computer Science, AMS*, pages 85–133, 1998.

[60] J. Matousek, R. Seidel, and E. Welzl. How to net a lot with little: Small epsilon-nets for disks and halfspaces. In *Proceedings of the 6th Annual Symposium of Symposium of Computation Geometry (SOCG)*, pages 16–22, 1990.

[61] J. McCabe. On serial files with relocatable records. *Operation Research*, 13:609–618, 1965.

[62] K. Mehlhorn. Data structures and algorithms 1: Sorting and searching. *EATCS Monographs on Theoretical Computer Science*, 1984.

[63] K. Mehlhorn, S. Näher, M. Seel, R. Seidel, T. Schilz, S. Schirra, and C. Uhrig. Checking geometric programs or verification of geometric structures. In *Proceedings of 12th Annual Symposium on Computational Geometry (SOCG)*, pages 159–165, 1996.

[64] M. Mihail. Conductance and convergence of markov chains-a combinatorial treatment of expanders. In *Proceedings of the 30st Annual Foundations of Computer Science (FOCS)*, pages 526–531, 1989.

[65] N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. In *Proceedings of 12th Annual Symposium on Discrete Algorithms (SODA)*, pages 439–447, 2001.

[66] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[67] A. Nachmias and A. Shapira. Testing the expansion of a graph. Technical Report TR07-118, ECCC, 2007.

[68] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. Technical report, ECCC, 2004.

[69] R. Rivest. On self-organizing sequential search heuristics. *Communcations of the ACM*, 19:63–67, 1976.

[70] D. Ron. Property testing. *Handbook on Randomization*, II:597–649, 2001.

[71] Dana Ron and Michal Parnas. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381:183–196, 2007.

[72] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal of Computing*, 25:647–668, 1996.

[73] Albers S. and M. Mitzenmacher. Average case analyses of list update algorithms. *Algorithmica*, 21:312–329, 1998.

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

[75] A. Sinclair. Improved bounds for mixing rates of markov chains and multicommodity flow. *Combinatorics, Probability & Computing*, 1:351–370, 1992.

[76] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[77] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32:652–686, 1985.

[78] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.