

Local Monotonicity Reconstruction*

Michael Saks[†]

C. Seshadhri

saks@math.rutgers.edu
Dept. of Mathematics
Rutgers University

csesha@cs.princeton.edu
Dept. of Computer Science
Princeton University

Abstract

We investigate the problem of monotonicity reconstruction, as defined by Ailon, Chazelle, Comandur and Liu (2004) in a localized setting. We have oracle access to a nonnegative real-valued function f defined on the domain $[n]^d = \{1, \dots, n\}^d$ (where d is viewed as a constant). We would like to closely approximate f by a monotone function g . This should be done by a procedure (a *filter*) that given as input a point $x \in [n]^d$ outputs the value of $g(x)$, and runs in time that is polylogarithmic in n . The procedure can (indeed must) be randomized, but we require that all of the randomness be specified in advance by a single short random seed. We construct such an implementation where the time and space per query is $(\log n)^{O(1)}$ and the size of the seed is polynomial in $\log n$ and d . Furthermore, with high probability, the ratio of the (Hamming) distance between g and f to the minimum possible Hamming distance between a monotone function and f is bounded above by a function of d (independent of n).

This allows for a local implementation: one can initialize many copies of the filter with the same short random seed, and they can autonomously handle queries, while producing outputs that are consistent with the same approximating function g .

*A preliminary version of this paper was titled “Parallel Monotonicity Reconstruction” [32].

[†]This work was supported in part by NSF under grants CCF-0515201 and CCF-0832787.

1 Introduction

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 effects of some sources of error, but often 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 but only if the structural property is maintained.

These considerations motivated the formulation of the *online property reconstruction* model, which was introduced in [3]. We are given a data set, which we think of as a function f defined on some domain Γ . 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 not too large 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). For this reason, [3] considered online reconstruction algorithms. Such an algorithm, called a *filter*, gets as input a sequence x_1, x_2, \dots of elements of Γ presented one at a time and must output the sequence of values $g(x_1), g(x_2), \dots$ where $g(x_i)$ is produced in response to x_i , before knowing x_{i+1} . The filter can access the function f via an oracle which given $y \in \Gamma$ answers $f(y)$. The aim is to design a filter which, given any permutation of Γ , outputs a function g satisfying (1) and (2) above and furthermore produces each successive $g(x_i)$ quickly, i.e., in time much smaller than $O(|\Gamma|)$.

In [3], a filter for the *monotonicity property* was given. In this setting, the domain Γ is the set $[n]^d = \{(j_1, \dots, j_d) : j_i \in [n]\}$, where $[n]$ denotes the set $\{1, 2, \dots, n\}$. The set $[n]^d$ is considered to be partially ordered under the component-wise (product) order: $(i_1, \dots, i_d) \leq (j_1, \dots, j_d)$ iff $\forall r, i_r \leq j_r$. A function f defined on Γ is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$. The filter they constructed satisfies Condition (1), has error blow-up that is bounded above by $2^{O(d)}$ (independent of n), and answers each successive query in time $(\log n)^{O(d)}$.

1.2 Local property reconstruction

The filter for monotonicity proposed in [3] has the following general structure. For each successive query x_j , the filter executes a randomized algorithm to compute $g(x_j)$. This algorithm accesses f , and also needs to access the answers $g(x_i)$ for $i < j$ to the queries asked previously. In particular,

the function g produced may depend on both the order of the queries as well as the random bits used by the algorithm.

This general structure for filters has two potential drawbacks: (1) It requires the storage of all previous queries and answers, thus incurring possibly significant space overhead for the algorithm, (2) It does not support a local implementation in which multiple copies of the filter, having read-only access to f are able to handle queries independently while maintaining mutual consistency.

In this paper, we propose the following strengthened requirements for a filter. A *local filter*¹ for reconstructing property \mathcal{P} is an algorithm A that has oracle access to a function f on domain Γ (the “data set”) 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 produce an output $A_{f,\rho}(x)$. We want A to satisfy the following properties:

1. For each f and ρ , $A_{f,\rho}$ satisfies \mathcal{P} .²
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. The size of the random seed ρ should be “much smaller” than $|\Gamma|$.

A local filter can be used, trivially, as an online filter. In addition, such a filter affords an obvious distributed implementation: generate one random seed, and give the same random seed to each of the copies of the filter. Since $A_{f,\rho}$ is deterministic all of the copies will behave identically.

For a local filter, there are three parameters of interest: the error blow-up, the time per query and the number of random bits needed for ρ (to initialize the filter). The memory used by a local filter is bounded by the sum of the length of ρ and the maximum time of a single query. By keeping these both small (e.g., much smaller than $|\Gamma|$) we obtain an online filter which uses little auxiliary space.

1.3 Related Work

One case of property reconstruction that has been studied extensively is *error correcting codes*. Suppose $C \subseteq \{0,1\}^n$ is such a code in which all members of C are pairwise at distance at least d . Let \mathcal{P} be the property of being a codeword. The error correction problem for C is to find the *closest codeword* to a given input string x . This can be formulated as a reconstruction problem for the property \mathcal{P} .

One variant of the error correction is the problem of local decoding. This problem was explicitly named in [27], but as noted there was studied previously in connection with self-correcting computation (e.g. [12, 21]) probabilistically checkable proofs (e.g. [7]), average-case reductions (e.g. [8, 33]) and private information retrieval (e.g. [13]). Here we want a decoding algorithm for a given code that, given oracle access to the bits of an input string x , and given an index $i \in [n]$, finds the i th bit of the closest codeword to x by querying a small (possibly randomly selected) number of bits of x . If we view the local decoding algorithm as a deterministic algorithm that takes as input i and

¹This was originally called a *parallel filter* in the conference version [32]. We made this terminology change since it is more compatible with the existing concepts of locally decodable codes.

²In an earlier version of this paper, this condition was replaced by the weaker condition that for each f , $A_{f,\rho}$ should satisfy \mathcal{P} with high probability. Prompted by a question raised by a referee we were able to modify our monotonicity filter to satisfy this stronger property, and so modified the definition accordingly. The weaker condition may be more appropriate for some other properties.

a random string r (used to make the decisions) then we require that for each i , most choices of r lead to the correct value for the i th bit of the closest codeword.

This is very similar to (though not quite the same as) the local property reconstruction problem for \mathcal{P} ; for local property reconstruction we interchange the “for all” and “for most” quantifiers and require that for most choices of r , and for all $i \in [n]$, the algorithm correctly produces the i th bit of the codeword. Also, we pay attention to the length of the random string r , which we want to be suitably small.

In local list decoding, our aim is to find a short *list* of codewords that are all suitably close to the input word. For example, in list-decoding of low-degree polynomials [6,33], the input is a function and the output is a *small list* of low-degree polynomials that are close to the input function.

The monotonicity problem considered in this paper is qualitatively quite different from the local decoding examples. In local decoding there is either *one* correct output, or (in the case of list-decoding) a *sparse list* of possible correct outputs. For monotonicity there may be many (possibly infinitely many) ways to correct a given function to a nearby function with the desired property. One might think that having many possible close corrections (rather than one) makes reconstruction easier but, at least for the monotonicity problem, it does not. The difficulty arises from the requirement that once the random seed is fixed, all query answers provided by the filter must be consistent with a *single* function having the property.

A related notion of reconstruction was discussed in [24], for generalized partition problems in dense graphs. Given an input dense graph G that satisfies some partition property (say k -colorability), we wish to efficiently construct a partition of the vertices that has at most an ε -fraction of violating edges. The algorithms for this provided in [24] behaved like local filters. Specifically, there was a constant (function of ε) time algorithm that gave the color class of an input vertex of G , and this could be run independently on all vertices (after fixing a random seed). This coloring was guaranteed to violate at most an ε -fraction of the edges in G .

The filter for monotonicity that we shall describe borrows techniques used for property testing of monotonicity. There has been a large amount of work done on property testing, which was defined in [24,31]. Many testers have been given for a wide variety of combinatorial, algebraic, and geometric problems (see surveys [17,22,30]). The related notions of tolerant testing and distance approximation were introduced in [29]. The problem of monotonicity in the context of property testing has been studied in [1,9,11,14,15,18,19,23,25]. Sublinear algorithms for *approximating* the distance of a function to monotonicity have been given in [2,16,29].

In general, a local filter for reconstructing a given property can be used to estimate the distance of an input instance to the property. When we fix a random seed and run the filter on f , the filter implicitly outputs a function g that has the desired property and is at distance at most $B\varepsilon_f$ from f (where ε_f is distance of f to \mathcal{P}). By choosing a random sample of domain points x and computing the fraction of points where $g(x) \neq f(x)$, we get an estimate of the distance $d(g, f)$ to any desired accuracy. Since $\varepsilon_f \leq d(g, f)$ and with high probability, $\varepsilon_f \geq d(g, f)/B$, we get a multiplicative B -approximation to ε_f in sublinear time.

1.4 Our results

In this paper, we construct a local filter for monotonicity for functions defined on $[n]^d$ 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 online filter for monotonicity of [3] has a running time per query of $(\log n)^{O(d)}$ (with a better constant in the exponent) and an error blow-up of 2^d . We see that our filter achieves local behavior while having query time and error blow-up that are similar to (but not quite as good) as those obtained by [3].

1.5 Why are local filters hard to design?

The starting point for the construction of our local filter for monotonicity is the online filter of [3]. We now give the main ideas of their construction, and indicate the difficulties in making their construction localized. 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 [3] is to classify the domain points as *accepted* and *rejected* in such a way that the following conditions hold -

- (1) There is a fast algorithm for testing whether a given point is accepted or rejected.
- (2) There are not many rejected points³.
- (3) The function restricted to the set of accepted points is monotone.

The third property ensures that it is possible (though not necessarily efficiently) to change the function only on rejected points and make the function monotone. To do this define $m(x)$ for $x \in \Gamma$, to be the largest accepted 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.

In [3], a point x is rejected 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 accepted/rejected classification there seems to be no fast way to compute $m(x)$. Instead, given query point x , the filter in [3] selects a sample of points less than or equal to x (called the *sample* of x), chooses $m'(x)$ to be the largest accepted point in the sample, and defines $g(x) = f(m'(x))$. The sampling procedure chooses $z < x$ with probability (roughly) inversely proportional to the distance of z from x ; in particular the sample includes x itself so that if x is accepted then $m'(x) = x$.

Defining g in this way creates a problem: g need not be monotone. For example, let y be a point and $x = m(y) < y$ be the largest accepted point less than or equal to y . Suppose that a query is made to y and x is not in the sample of y , so $m'(y) < x$. Suppose further that $f(m'(y)) < f(x)$. Suppose that after setting $g(y)$ to $f(m'(y))$, a query is made to index x . Since x is an accepted point we will have $m'(x) = x$ and so $g(x) = f(x)$, but this will violate monotonicity with the already defined $g(y) = f(m'(y))$.

In online reconstruction, this is not a significant problem because 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 [3].

Local reconstruction does not have this luxury. What we do is to redefine $m'(x)$ so as to guarantee that for any $y > x$, we have $m'(y) \geq m'(x)$. To do this, after sampling the points less than x we identify certain points of the sample which have the potential for creating non-monotonicities and exclude them from the sample. For example, in the scenario above, The point x needs to be excluded from its own sample to avoid the *potential* non-monotonicity with y . Notice that when we exclude x from its own sample we may introduce a new point where $g(x) \neq f(x)$, so we can't do this too often.

Thus, the main challenge in designing a local filter is to find an efficient way to identify the points that need to be excluded from the sample of x to avoid potential non-monotonicities with points greater than x , while not excluding x from its own sample too often.

³The number of rejected points is comparable to the distance of f to monotonicity.

The difficulties in designing a local filter are 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 [3] this is done by finding a polylogarithmic size sample that is a suitable approximation to $M(x)$, and then defining $g(x)$ to be the maximum of $f(y)$ for y in the sample.

As with the one-dimensional case, using an approximation to $M(x)$ destroys the guarantee that 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 local filter can't save these values, we again need to judiciously exclude points from the sample to avoid non-monotonicities.

The definition of the sample, which we denote $\text{REP}(x)$, crucially uses a data structure of nested boxes (products of intervals). Ensuring Condition (3) is accomplished by a careful and efficient scheme for passing crucial information about the distribution of rejected points in a particular box to its sub-boxes.

1.6 Preliminaries

We consider data sets as functions defined on a fixed finite domain Γ . A property \mathcal{P} is a collection of such functions. The distance $d(f, g)$ between two functions f and g is the fraction of $x \in \Gamma$ for which $f(x) \neq g(x)$. The distance of function f to property \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 $\{1, 2, \dots, m\}$. Throughout this paper, $\Gamma = [n]^d = \{(x_1, \dots, x_d) : \forall i \in [d], x_i \in [n]\}$ 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. The domain 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*. For $x, y \in \Gamma$, the interval $[x, y]$ is the set of points $\{z | x \leq z \leq y\}$. Points are generally denoted by lower case letters, sets of points are denoted by upper case letters and sets of intervals of points by calligraphic letters.

The subset of Γ consisting of points with at least one coordinate equal to 1 is called the *lower boundary* of Γ . Points with all coordinates at least 2 are called *non-lower-boundary* points.

We consider functions mapping Γ to the nonnegative reals. For convenience of presentation, we assume that our functions have value 0 on the lower boundary. This assumption is without significant loss of generality. Assume we have a filter that works for functions satisfying this restriction. Suppose we are given an arbitrary nonnegative function f on $[n]^d$ for which we wish to obtain a suitable monotone correction g . Consider the function f' on $[n+1]^d$ that has value 0 on its lower boundary, and has value $f'(x) = f(x_1 - 1, \dots, x_n - 1)$ for x not on the lower boundary. One can think of f' as the function f with an extra padding of zeroes at the lower boundary. Because this padding of zeroes does not add any violations to monotonicity, $\varepsilon_f n^d = \varepsilon_{f'} (n+1)^d$ (where ε_f and $\varepsilon_{f'}$ are the respective distances to monotonicity). We can run our filter on the function f' instead of f . For $x \in [n]^d$, we define $g(x)$ to be the output of the filter for f' applied to the point $(x_1 + 1, \dots, x_d + 1)$.

We need this assumption about the lower boundary because our filter outputs value 0 for every

point on the lower boundary. This can be avoided by a somewhat tedious modification of the algorithm that would introduce some special cases. Since this adds nothing to our filter and only complicates the exposition, we have decided to impose this assumption.

As defined in the introduction, a local 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 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 $|\Gamma|$. Similarly, a high probability event is one having probability $1 - (1/|\Gamma|^{\omega(1)})$.

2 A high level view of the filter

The running times given below are *not* sample complexities (which count only the number of values of f that are seen) but account for the all the computation done.

The first component of the filter is a function SIFT, which takes as input a point $x \in [n]^d$ and returns *accept* or *reject*. We define the subsets *Accepted* and *Rejected* of $[n]^d$ in the natural way. Given a function f on $[n]^d$, we say that a subset $S \subseteq [n]^d$ is *f-admissible* if the restriction of f to S is monotone. The function SIFT satisfies:

S1 : *Accepted* is *f*-admissible.

S2 : With high probability, $|Rejected| \leq C_1(d)\varepsilon_f n^d$ where $C_1(d)$ is independent of n . (For our filter, we will have $C_1(d) \leq \frac{1}{9}(10^{d+1} - 10)$.)

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

The second part of the filter is a function REP that takes as input a point $x \in [n]^d$ and (using the subroutine SIFT) returns a set $REP(x)$ of *representative points* for x . This subroutine satisfies:

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

B2 : $REP(x) \subseteq Accepted$.

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 non-lower-boundary points x for which $x \notin REP(x)$ is at most $C_2(d)|Rejected|$, where $C_2(d)$ is independent of n . (For our filter, $C_2(d) = 2^{O(d^2)}$.)

B5 : REP runs in time $(\log n)^{O(d)} \times$ the running time of SIFT.

On input x our filter outputs:

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

Let us see that properties [S1]-[S3] and [B1]-[B5] ensure that the filter has the required properties (Conditions (1)-(4) in Section 1.2). To see that g is monotone, let $x \leq y$. If $g(x) = 0$, then $g(x) \leq g(y)$. So let us assume that $g(x) > 0$. 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 $z, z' \in Accepted$, so by [S1], $f(z) \leq f(z')$. By the definition of $g(y)$ we have $g(y) \geq f(z')$ and so $g(y) \geq f(z') \geq f(z) = g(x)$ as required.

We now provide an upper bound on the number of points for which $g(x) \neq f(x)$. Note that $x \in REP(x)$ or x on the lower boundary implies $g(x) = f(x)$. By [B4], the number of points with $g(x) \neq f(x)$ is at most $C_2(d)|Rejected|$ 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].

3 The one-dimensional case

The details of the functions SIFT and REP are considerably simpler for the one-dimensional case, so we begin with that. We need some definitions.

3.1 The set \mathcal{I} of intervals

For $x, y \in [n]$ the *interval* $[x, y]$ is the set $\{z \in [n] : x \leq z \leq y\}$. For subset J , interval I and point x we define:

- $\min(J)$ denotes the least element of J .
- $\max(J)$ denotes the greatest element of J .
- $\text{Span}(J)$ is the unique smallest interval containing J , i.e., the interval $[\min(J), \max(J)]$.
- If $\min(I) = 1$, we say I is *left-extreme* and if $\max(I) = n$ we say I is *right-extreme*.
- I is *left of* x if $\max(I) \leq x$,
- I is *right of* x if $\min(I) \geq x$.
- We say I is *near to* x if $x \notin I - \{\min(I), \max(I)\}$ and $|\text{Span}(I \cup x)| < 2|I|$.
- I is *left-near* (resp., *right-near*) to x if it is left (resp., right) of x and near to x . Observe that if I is near to x , then since $x \notin I - \{\min(I), \max(I)\}$ I is left-near or right-near to x .

Our filter makes use of a carefully chosen set \mathcal{I} of intervals of n . As stated earlier, we assume $n = 2^k$ for some integer k .

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 *i th level*, is the set of intervals $\{I_i^j | j \geq 0\}$. The *i th level* contains $\frac{n}{2^{i-1}} - 1$ intervals each of size 2^i . Notice that the intervals in the first level are size 2 (and not size 1).

An interval I_i^j is said to be *even* if j is even and *odd* if j is odd. 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 with vertex set \mathcal{I} , and edge set consisting of pairs (I, J) where $J \subseteq I$ and $|I| = 2|J|$. The DAG D has $\log n$ levels and $2n - \log n - 2$ vertices ($n/2^{i-1} - 1$ at level i). The interval $I_{\log n}^0 = [1, n]$ 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 1 shows the first three levels of the DAG D (with edges pointing downwards). We also show the intervals of each level. The root level has a single interval, which is the whole domain. The next level has three overlapping intervals of the same size, and the third level has seven overlapping intervals.

Suppose I and J are intervals and there is an edge from I to J . We say that I is a *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 .

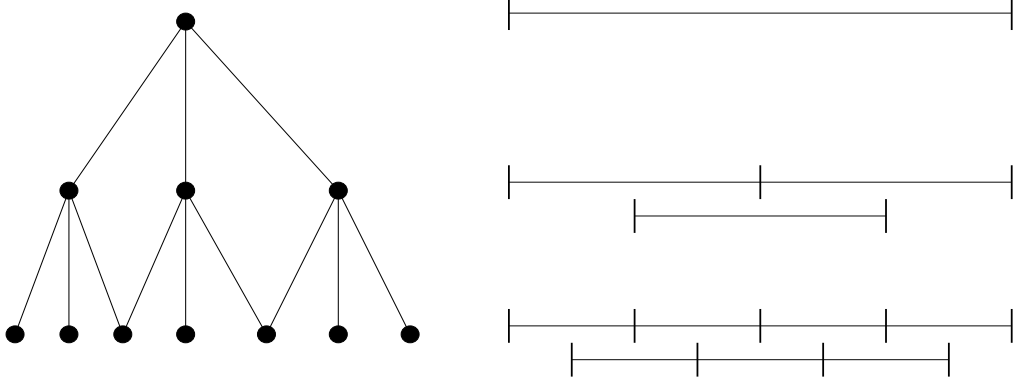


Figure 1: *The DAG D*

- $\max(I) = \max(J)$. 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.

The family \mathcal{I} was chosen to satisfy certain properties which we now state and prove:

- I1:** Every interval in \mathcal{I} has size 2^i for some $1 \leq i \leq d$.
- I2:** For each $x \in [n]$, and $j \in [\log n]$ there are at most 2 intervals in \mathcal{I} of size 2^j that are left-near and at most 2 intervals of size 2^j that are right-near to x . Thus there are at most $2 \log n$ intervals of \mathcal{I} that are left-near to x and at most $2 \log n$ intervals that are right-near to x .
- I3:** For any two points $x < y$, there exists an interval $I \in \mathcal{I}$ such that $I \subseteq [x, y]$ and I is near to both x and y .
- I4:** For each interval $I \in \mathcal{I}$ that is not right-extreme, there is a unique interval called the *non-left parent* of I , denoted $NonLeftPar(I)$, that satisfies:
- $I \subseteq NonLeftPar(I)$,
 - $|NonLeftPar(I)| = 2|I|$,
 - $\max(I) < \max(NonLeftPar(I))$.

Thus from I there is a unique sequence $I = I_1, I_2, \dots, I_r$, called the *non-left path from I* such that for $1 \leq j \leq r - 1$, I_{j+1} is the non-left parent of I_j and I_r is right-extreme. The intervals on the non-left path from I (including I itself) are the *non-left ancestors* of I .

- I5:** If $J \neq I$ is a non-left ancestor of I then I is disjoint from $Upper(J)$ where $Upper(J)$ denotes the rightmost $|J|/4$ points of J (which is well-defined since $|J| \geq 4$ is a power of 2).

I6: If I is to the left of x , then some non-left ancestor of I (possibly I itself) is left-near to x .

We verify properties [I1-I6]. Property [I1] is immediate.

For [I2], we note that if I is an interval of size 2^k that is left-near to x then $\max(I) \in [x - 2^k + 1, x]$. Since the right endpoints of intervals at level k are spaced 2^{k-1} apart, at most two such endpoints lie in $[x - 2^k + 1, x]$, so at most two intervals at level k are left-near to x . A similar argument shows that at most two intervals at level k are right-near to x .

For [I3], let k be the largest integer such that $2^k \leq |[x, y]|$. Let w be the unique multiple of 2^{k-1} satisfying $0 \leq w + 1 - x < 2^{k-1}$. Let $I_1 = [w + 1, w + 2^{k-1}]$ and $I_2 = [w + 1, w + 2^k]$. Then I_1 belongs to \mathcal{I} and is right-near to x , and left of y , since $y \geq x + 2^k \geq w + 1 + 2^{k-1}$. If I_1 is left-near to y , we are done. Suppose I_1 is not left-near to y . Then $y \geq \max(I_1) + |I_1| = \max(I_2)$ so I_2 is left of y and $I_2 \in \mathcal{I}$. Since $y - x + 1 = |[x, y]| < 2^{k+1}$, we have $y < x + 2^{k+1} - 1 \leq (w + 2^k) + 2^k = \max(I_2) + |I_2|$, and thus I_2 is left-near to y .

For [I4], suppose that I is not right-extreme. If I is odd, it has one parent, its central parent, which satisfies the three stated conditions. If I is even, then its right parent satisfies the three conditions. For [I5], suppose I has size 2^i and J' is *NonLeftPar*(I). Since $\max(J) = \max(I) + 2^{i-1}$ and J' has size 2^{i+1} , *Upper*(J) is disjoint from I . Applying this argument repeatedly along the non-left path from I , we get that *Upper*(J) is disjoint from I for any non-left ancestor J .

For [I6], let J be the largest non-left ancestor of I such that $\max(J) \leq x$. If J is right-extreme then $x = n$ is the right endpoint of J so J is near x . Otherwise, we have $x < \max(K)$ where K is the non-left parent of J , and so *Span*($J \cup x$) is a proper subset of K and so has size at most $|K| - 1 = 2|J| - 1$, so J is left-near to x .

3.2 SIFT and REP in one dimension

We are now ready to describe SIFT and REP. As described in Section 2, this is enough to completely specify our filter.

The filter takes as input a random seed, which consists of $2t \log n$ bits where $t = c(\log n)^2$ for some sufficiently large constant c . (The size of t is chosen to make Lemmas 4.1 and 4.3 true.) We interpret the random seed as a pair of sequences $(s(1), \dots, s(t))$ and $(r(1), \dots, r(t))$ with elements from $[n]$.

We use the random seed to define procedures *SAMPLE*₁ and *SAMPLE*₂ each taking as input an interval I from \mathcal{I} and returning a sequence of elements from I of length $\min(|I|, t)$. We first define *SAMPLE* which takes an interval I and a sequence $w = (w(1), \dots, w(m))$ with integer entries.

SAMPLE(I, w) outputs a sequence of length $\min(|I|, |w|)$ with entries from I . If $|w| \geq |I|$ then *SAMPLE*(I, w) has length $|I|$ and is the sequence of elements of I in increasing order. If $|w| < |I|$ then *SAMPLE*(I, w) has length $|w|$ and has i th entry $w(i) \bmod I$, which is defined to be the unique member m of I such that $w(i) - m$ is divisible by $|I|$.

*SAMPLE*₁(I) is defined to be *SAMPLE*(I, s).

*SAMPLE*₂(I) is defined to be *SAMPLE*(I, r).

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

SIFT(x): *Reject* x if there is an $I \in \mathcal{I}$ that is near to x , such that at least half of the elements of *SAMPLE*₁(I) are in violation with x , otherwise *accept* x .

The function $\text{REP}(x)$ constructs a set of elements less than or equal to x that are all accepted by SIFT. It makes use of two procedures that take as input an interval I . The procedure REFINE returns a subset of $\text{SAMPLE}_2(I)$, and the procedure $\text{UNSAFE}(I)$ classifies the interval I as *Unsafe* or *Safe*. $\text{UNSAFE}(I)$ is determined based on $\text{REFINE}(I)$ and $\text{REFINE}(I)$ depends on $\text{UNSAFE}(J)$ where J is the non-left parent of I .

$\text{REFINE}(I)$: If I is right-extreme, or if the non-left parent of I is *Safe* then $\text{REFINE}(I)$ consists of all points of $\text{SAMPLE}_2(I)$ that are accepted by SIFT. Otherwise (i.e., if the non-left parent of I is *Unsafe*), $\text{REFINE}(I) = \emptyset$.

$\text{UNSAFE}(I)$: Classify I as *Unsafe* if no point of $\text{REFINE}(J) \cap \text{Upper}(J)$ is accepted by SIFT, otherwise classify I as *Safe*.

$\text{REP}(x)$: This returns the union of $\text{REFINE}(I)$ over all intervals I that are left-near to x .

The above description of REFINE and UNSAFE is recursive; here is an alternative iterative implementation. In what follows, when we say that the algorithm computes $S \cap \text{Accepted}$ for some set S , we mean that the algorithm applies SIFT to each element of S and returns the subset of S that is accepted by SIFT. To evaluate $\text{REFINE}(I)$ and $\text{UNSAFE}(I)$ consider the non-left path $I = I_1, I_2, \dots, I_r$ where I_r is right-extreme. Then $\text{REFINE}(I_r) = \text{SAMPLE}_2(I_r) \cap \text{Accepted}$ and I_r is *Unsafe* if $\text{REFINE}(I_r) \cap \text{Upper}(I_r) = \emptyset$. For $j < r$, having determined whether I_{j+1} is *Unsafe*, we set $\text{REFINE}(I_j) = \emptyset$ if I_{j+1} is *Unsafe* and to be $\text{SAMPLE}_2(I_j) \cap \text{Accepted}$ otherwise. Also we declare I_j to be *Unsafe* if no points of $\text{REFINE}(I_j) \cap \text{Upper}(I_j)$ are accepted by SIFT and to be *Safe* otherwise.

4 Proof of correctness of one-dimensional SIFT and REP.

It now suffices to verify that SIFT and REP satisfy conditions [S1-S3] and [B1-B5] from Section 2.

[S1]: Suppose x, y is a violation with $x < y$. By property [I3], there is an $I \in \mathcal{I}$ with $I \subseteq [x, y]$ that is near to both x and y . Since x, y is a violation, for each $z \in I \subseteq [x, y]$, z is in violation with at least one of x and y . Therefore one of x and y is in violation with at least half the elements of $\text{SAMPLE}_1(I)$, and so x will be rejected by SIFT. Note that this is true with probability 1.

[S2]: We say SAMPLE_1 fails for (x, I) , where $x \in [n]$ and I is near x , if x is in violation with fewer than $.4|I|$ elements of I but is in violation with at least half the elements of $\text{SAMPLE}_1(I)$. We say that SAMPLE_1 fails if there is an $x \in [n]$ and I near x such that SAMPLE_1 fails for x and I .

Lemma 4.1 *The probability that SAMPLE_1 fails is $e^{-\Omega(t)} n \log n = n^{-\omega(1)}$.*

Proof: Fix (x, I) and consider the event that SAMPLE_1 fails for (x, I) . If $|I| < t$ then the fraction of points of $\text{SAMPLE}_1(I)$ that are in violation with x is equal to the fraction of points of I that are in violation with x , and so SAMPLE_1 must succeed for (x, I) . If $|I| > t$ then $\text{SAMPLE}_1(I)$ is a sequence of length t , whose elements are independently and uniformly chosen from I . Thus the number of elements in $\text{SAMPLE}_1(I)$ that are in violation with x is binomially distributed with parameter at most 0.4. By standard tail estimates for the binomial distribution (such as the Chernoff bound), the probability that the fraction of points in $\text{SAMPLE}_1(I)$ in violation with x is at least .5 is $e^{-\Omega(t)}$. By [I2] there are at most $4n \log n$ pairs (x, I) with I near x . By a union bound, the probability that SAMPLE_1 fails is at most the sum over all such pairs of the probability that SAMPLE_1 fails for (x, I) . Thus, we get an upper bound of $4n \log n e^{-\Omega(t)}$, which is $n^{-\omega(1)}$ for $t = c(\log n)^2$, as chosen at the

beginning of Section 3.2. □

We now show that if SAMPLE_1 does not fail, then SIFT rejects at most $10\varepsilon_f n$ points. Let Q be a set of minimum size such that $[n] - Q$ is f -admissible; thus $|Q| = \varepsilon_f n$.

We claim that every rejected point x belongs to an interval J_x (not necessarily in \mathcal{I}) that contains at least $.2|J_x|$ points of Q . Suppose x is rejected. If $x \in Q$, we can take $J_x = \{x\}$. Suppose $x \notin Q$. Since x is rejected, there is an interval I near x such that x is in violation with at least half the points of $\text{SAMPLE}_1(I)$. Since SAMPLE_1 does not fail, x is in violation with at least $.4|I|$ points of I . Since $[n] - Q$ is f -admissible, x can only be in violation with points of Q , so $|I \cap Q| \geq .4|I|$. Let $J_x = \text{Span}(I \cup x)$; since I_x is near x , $|J_x| \leq 2|I_x|$ which implies that $|J_x \cap Q| \geq .2|J_x|$.

If S, I are sets and $\theta \in [0, 1]$ we say that S is θ -dense in I if $|S \cap I| \geq \theta|I|$.

Lemma 4.2 *Let $S \subseteq [n]$ and $\theta > 0$ be a real number. Let \mathcal{C} be any family of intervals such that S is θ -dense in each member of \mathcal{C} . Then $|\bigcup_{I \in \mathcal{C}} I| \leq \frac{2}{\theta}|S|$.*

Proof: We may assume without loss of generality that no subcollection of \mathcal{C} has the same union as \mathcal{C} , otherwise we replace \mathcal{C} by the subcollection. There can't be three intervals that cover the same element z , since then one of them is in the union of the other two and can be omitted.

Then (with all sums and unions taken over $I \in \mathcal{C}$) we have

$$|(\bigcup_I I) \cap S| \geq \frac{1}{2} \sum_I |I \cap S| \geq \frac{\theta}{2} \sum_I |I| \geq \frac{\theta}{2} |\bigcup_I I|,$$

as required to prove the lemma. □

Applying this lemma to the collection $\{J_x : x \in \text{Rejected}\}$ with $S = Q$ and $\theta = .2$ we conclude that:

$$|\text{Rejected}| \leq \left| \bigcup_{x \in \text{Rejected}} J_x \right| \leq 10 \left| Q \cap \left(\bigcup_{x \in \text{Rejected}} J_x \right) \right| \leq 10|Q| = 10\varepsilon_f n,$$

to complete the proof of [S2].

[S3]: Computing $\text{SIFT}(x)$ involves examining at most $4 \log n$ intervals near x . For each such interval I , one must look at the at most $c(\log n)^2$ points of $\text{Sample}_1(I)$ to see if it is a violation. The overall running time is $O((\log n)^3)$.

[B1-B2]: By definition of the algorithm, $\text{REP}(x) \subseteq \text{Accepted}$ and only includes points that lie in intervals that are left-near to x .

[B3]: Let x, y be arbitrary points in $[n]$ with $x \leq y$ and let $z \in \text{REP}(x)$. We must show that there is a $z' \in \text{REP}(y)$ with $z \leq z'$. Since $z \in \text{REP}(x)$, there is an interval I that is left-near x such that $z \in \text{REFINE}(I)$.

Let $I = I_1, I_2, \dots, I_w$ be the non-left path from I . We claim that I_i is safe for all $i \in [2, w]$. Suppose not, and let j be the least index in $[2, w]$ such that I_j is *Unsafe*. Then $\text{REFINE}(I_{j-1}) = \emptyset$ and so I_{j-1} is *Unsafe*, and so by the choice of j , $j = 2$. But then $\text{REFINE}(I_1) = \emptyset$ contradicts that $z \in \text{REFINE}(I)$.

By [I6], for some $r \in [w]$, I_r is left-near to y . If $r = 1$ we have $z \in \text{REFINE}(I_r) \subseteq \text{REP}(y)$ so we can take $z' = z$. Otherwise, from above we have I_r is *Safe* and so $\text{REFINE}(I_r) \cap \text{Upper}(I_r)$ is non-empty. By [I5], $\text{Upper}(I_r)$ is disjoint from (and necessarily to the right of) I and so we can take

z' to be any point of $\text{REFINE}(I_r) \cap \text{Upper}(I_r)$.

[B4]: We bound the number of non-lower-boundary points for which $x \notin \text{REP}(x)$. We say that SAMPLE_2 fails for interval I if at least half the points of $\text{Upper}(I)$ are accepted by SIFT , but no points in $\text{SAMPLE}_2(I) \cap \text{Upper}(I)$ are accepted by SIFT . Otherwise, we say that SAMPLE_2 succeeds for I , which means that if at least $|I|/8$ points of $\text{Upper}(I)$ are accepted by SIFT , then $\text{SIFT}(\text{SAMPLE}_2(I) \cap \text{Upper}(I)) \neq \emptyset$.

We say that SAMPLE_2 succeeds if it succeeds for all intervals $I \in \mathcal{I}$ of size at least 4, and that SAMPLE_2 fails otherwise.

We will shortly prove that SAMPLE_2 succeeds with high probability. Assuming that SAMPLE_2 succeeds, we show that the number of non-lower-boundary points x such that $x \notin \text{REP}(x)$ is at most $16|\text{Rejected}|$. By Lemma 4.2, it suffices to show that each such x belongs to an interval I_x for which at least $1/8$ of the points belong to Rejected .

If $x \in \text{Rejected}$, we take $I_x = \{x\}$. So assume $x \in \text{Accepted}$. We have $x \geq 2$ since x is a non-lower-boundary point. The interval $[x-1, x]$ is left-near to x , so if $x \notin \text{REP}(x)$ then $x \notin \text{REFINE}([x-1, x])$. Since $x \in \text{Accepted} - \text{REFINE}([x-1, x])$, the non-left parent of $[x-1, x]$ must be *Unsafe*. Define I_x to be the largest non-left ancestor of $[x-1, x]$ that is *Unsafe*. Then $\text{SIFT}(\text{SAMPLE}_2(I_x)) \cap \text{Upper}(I_x) = \emptyset$. Since SAMPLE_2 succeeds on I_x , at least half the points in $\text{Upper}(I_x)$ are in Rejected , which implies that at least $1/8$ of the points of I_x belong to Rejected , as required.

Lemma 4.3 SAMPLE_2 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_2 fails for a fixed interval I . If fewer than half the points of $\text{Upper}(I)$ are accepted by SIFT , then SAMPLE_2 succeeds on I . So assume that at least half the points of $\text{Upper}(I)$ are accepted by SIFT . If $t \geq |I|$ then $\text{SAMPLE}_2(I)$ is just the sequence of points in I and so includes points in $\text{Upper}(I) \cap \text{Accepted}$. If $t < |I|$ then $\text{SAMPLE}_2(I)$ is a sequence of elements chosen independently and uniformly from I . Each element has probability at least $1/8$ of being in $\text{Upper}(I) \cap \text{Accepted}$. Therefore, the probability that no points of $\text{SAMPLE}_2(I)$ are in $\text{Upper}(I) \cap \text{Accepted}$ is at most $(7/8)^t$. Taking a union bound over all intervals in \mathcal{I} yields an upper bound of $O(n \log n)(7/8)^t = n^{-\omega(1)}$ on the probability that SAMPLE_2 succeeds.

Then, for each point in $\text{SAMPLE}_2(I)$, the probability that it is one of the points in $\text{Upper}(I)$ accepted by SIFT is at least $1/8$. By the independence of the samples in $\text{SAMPLE}_2(I)$, we can apply a multiplicative Chernoff bound to show that $\Pr[\text{SIFT}(\text{SAMPLE}_2(I) \cap \text{Upper}(I)) = \emptyset] = 2^{-\Omega(t)}$. A union bound over all intervals and points completes the proof. \square

[B5]: The running time of REP is bounded as follows. For any x , there are at most $2 \log n$ intervals I (at most 2 on each level) that are left-near with respect to x . For each such I , we compute $\text{REFINE}(I)$ which involves computing $\text{SAMPLE}_2(J)$ for each of the at most $\log n$ non-left ancestors J of I and calling SIFT for each of the points in $\text{SAMPLE}_2(J)$. Since the size of $\text{SAMPLE}_2(J)$ is $t = O(\log^2 n)$ the running time of REP is at most the cost of $O((\log n)^4)$ calls to SIFT . (The exponent of $\log n$ can probably be improved.)

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

Remark. (Partial Functions) The algorithms for the one-dimensional case extend to partial functions f , in which for some points x , $f(x)$ is unassigned. We define ε_f for partial functions f to be the minimum fraction of points that must be changed or assigned to make f a monotone total function. We extend the definition of a violation to say that x, y is a violation if either $f(x)$ or

$f(y)$ is unassigned or $x < y$ and $f(x) > f(y)$. With these definitions, it is easy to see that SIFT will reject all x such that $f(x)$ is unassigned, and that the proofs that SIFT and REP satisfy the needed properties go through as above. This generalization will be needed for the definition of the multidimensional version of SIFT in Section 5.4

5 A filter for multidimensional data

5.1 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 $B = B_1 \times \cdots \times B_d$ where $B_r = [x_r, y_r]$ is the interval obtained by projecting B onto the r th 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 $x^{(r)}$.

For any r -line L , there is a natural bijection between L and $[n]$ that maps $x \in L$ to $x_r \in [n]$. 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 . A subset of L corresponding to an interval is called an L -*interval*. We say that J is a *line interval* if it is an L -interval for some line L . Define $\mathcal{I}[L]$ to be the set of $I[L]$ for $I \in \mathcal{I}$, where $\mathcal{I} = \mathcal{I}(k)$ is as defined in Subsection 1.6.

5.2 Intuition for the multi-dimensional filter

It is easy to see that a (total) function f on $[n]^d$ is monotone if and only if it is monotone along every line; that is, if there is a violation, then there is a violation where both points lie on a common r -line for some r . Given that we already have a one-dimensional filter, it is natural to try a multi-dimensional filter based on recursion or induction, with the one-dimensional filter as the base case. For example, one can try the following approach: Apply the one dimensional filter separately to each line in direction 1. This ensures that there are no violations on any 1-line. Then apply the one dimensional filter separately to each line in direction 2, then to each line in direction 3, etc. The hope is that when we work on one direction we do not disrupt the monotonicity of lines in the directions previously processed. Unfortunately this is not true. Another inductive approach would be to assume that we have a filter for dimension $d - 1$ and construct one for dimension d . We were unable to make these ideas work.

We turn to the general approach based on SIFT and REP as outlined in Section 2. For SIFT, whose goal is to identify the points whose function values are to be changed without determining the new values, it turns out that a recursive approach based on the 1-dimensional SIFT does work. This approach is based on that used in [1, 3]. We define $\text{SIFT}_1, \text{SIFT}_2, \dots, \text{SIFT}_d$ where SIFT_j is defined recursively from SIFT_{j-1} and has the property that the set Rejected_j of points rejected by SIFT_j contains one point from any violation x, y whose points differ only in their first j coordinates. Having defined SIFT_{j-1} , let f_j be the partial function which agrees with f on Accepted_j and is unassigned on Rejected_j . SIFT_j is obtained by applying the one-dimensional SIFT to the partial function f_{j-1} (see the remark at the end of Section 4) separately along each j -line.

The d -dimensional version of REP is not inductive, but is obtained by generalizing the one-dimensional approach. The family \mathcal{I} of intervals is replaced by the family \mathcal{B} of d -dimensional boxes of the form $B = B_1 \times \cdots \times B_d$ where each $B_j \in \mathcal{I}$. The notion of a box being left-near a point

is obtained as a straightforward generalization of the one-dimensional notion. For each box B , we construct a small subset $\text{REFINE}(B)$ and take $\text{REP}(x)$ to be the union of $\text{REFINE}(B)$ over all boxes B that are left-near to x .

To define $\text{REFINE}(B)$, we start with a random sample $\text{BOXSAMPLE}(B)$ of points in B , as in the one-dimensional case. For a subtle technical reason (Observation 5.1), $\text{BOXSAMPLE}(B)$ is not a sequence of points chosen uniformly at random from B . Instead, it is obtained by taking small random samples from each of the intervals B_1, B_2, \dots, B_d and then taking the Cartesian products of these samples.

Recall that in the one-dimensional case we classified intervals as *Safe* or *Unsafe* and defined $\text{REFINE}(B)$ to be $\text{SAMPLE}(B) \cap \text{Accepted}$ if its non-left parent was *Safe*, and to be the empty set otherwise. The purpose of this was to avoid the violation of condition [B3]. In the d -dimensional case there is a straightforward generalization of the notions of non-left parent of a box (where a box may have one non-left parent in each direction). Based on the one-dimensional case, one can define $\text{REFINE}(B)$ to be $\text{BOXSAMPLE}(B) \cap \text{Accepted}$ or \emptyset depending on whether all non-left parents are *Safe* for some appropriate generalization of *Safe*. However, it seems that any generalization of *Safe* to boxes that declares enough points *Unsafe* so as to ensure that [B3] holds, would result in discarding too many points when evaluating $\text{REFINE}(x)$, thus violating [B4].

So we need a more careful definition of $\text{REFINE}(B)$, one that discards many fewer points from $\text{BOXSAMPLE}(B)$. This is the main complication in the algorithm and proof compared to the one-dimensional case. Safety is no longer just a Yes-No property of boxes. Instead $\text{UNSAFELINES}(B)$ returns a set of lines that meet B and are, in a precise sense, unsafe for B . The definitions of $\text{REFINE}(B)$ and $\text{UNSAFELINES}(B)$ are inductively intertwined: $\text{REFINE}(B)$ is obtained by removing from $\text{BOXSAMPLE}(B) \cap \text{Accepted}$ all points that lie on lines L that are unsafe for some non-left parent of B . Then the lines that are unsafe for B are determined based on $\text{REFINE}(B)$.

This somewhat complicated definition is intended to ensure that REP satisfies [B3], while avoiding removing too many points. Verifying [B4], turns out to be the hardest part, which we'll discuss further when we get to it.

5.3 The family \mathcal{B} of boxes

Before describing the function REP , we need some additional definitions.

We consider the set $\mathcal{B} = \mathcal{B}(k)_d$ to be the set of all boxes of the form $B = B_1 \times \dots \times B_d$ where each $B_j \in \mathcal{I}(k)$ (where $\mathcal{I}(k)$ was defined in Section 3.1). 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 \sim_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: $B \longrightarrow C$ in Δ , if and only if for some $r \in [d]$, $B \sim_r C$ and $B_r \longrightarrow C_r$ in the DAG D defined on \mathcal{I} in 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 . Note that each box has at most one non-left r -parent for each 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-near to* x if for each $j \in [d]$, B_j is left-near to x_j .

We say that a box C is a *non-left ancestor* of B if for each $j \in [d]$, C_j is a non-left ancestor of B_j with respect to the one-dimensional dag D . Between C and B there is at least one path $C = C_1, C_2, \dots, C_w = B$ where for $j \in [w-1]$, C_{j+1} is a non-left parent of C_j . 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.

5.4 SIFT and REP for d dimensions

We now specify the functions SIFT and REP for the d -dimensional case. As described in Section 2 this is enough to complete the description of the filter.

The random seed consists of $2dt \log n$ elements of $[n]$ where $t = c(\log n)^2$ for some sufficiently large constant c (independent of n and d). The size of t is chosen to imply the $n^{-\omega(1)}$ upper bound on failure in the proofs of [S2] in Section 5.5 and of [B3] in Section 5.6. We interpret the random seed as $2d$ sequences $s^1, s^2, \dots, s^d, r^1, \dots, r^d$, each consisting of t uniformly random elements of $[n]$.

We use the random seed to define d functions

$$\text{SAMPLE}_1^1, \dots, \text{SAMPLE}_1^d, \text{SAMPLE}_2^1, \dots, \text{SAMPLE}_2^d,$$

each taking as input an interval I from \mathcal{I} and returning a sequence of elements from I of length $\min(|I|, t)$. We use the same procedure SAMPLE as in the one-dimensional case, which takes an interval I and a sequence w with integer entries.

SAMPLE(I, w) outputs a sequence of length $\min(|I|, |w|)$ with entries from I . If $|w| \geq |I|$ then SAMPLE(I, w) has length $|I|$ and is the sequence of elements of I in increasing order. If $|w| < |I|$ then SAMPLE(I, w) has length $|w|$ and has i th entry $w(i) \bmod I$.

For $j \in [d]$, $\text{SAMPLE}_1^j(I)$ is defined to be SAMPLE(I, s^j).

For $j \in [d]$, $\text{SAMPLE}_2^j(I)$ is defined to be SAMPLE(I, r^j).

Our multi-dimensional SIFT will use the one-dimensional SIFT as a subroutine. To distinguish them, we will rename the 1-dimensional version as ONEDIMSIFT. Recall that ONEDIMSIFT has an explicit parameter $x \in [n]$ but also depends implicitly on the function f and on the interval sampling function SAMPLE₁. It will be convenient now to make this dependence explicit using the notation ONEDIMSIFT($x; f, \text{SAMPLE}_1$).

We now define a procedure LINESIFT based on ONEDIMSIFT. LINESIFT takes parameters $(x, j; h)$ where $x \in [n]^d$, direction $j \in [d]$, and h is a function on $[n]^d$ given by query access.

LINESIFT($x, j; h$). Let L be the line through x in direction j and let $h[L]$ be the restriction of h to L .

View L as a copy of $[n]$ via the map that takes $y \in L$ to y_j . Run ONEDIMSIFT($x; h[L], \text{SAMPLE}_1$).

We now recursively define for $j \in [d]$, partial functions f_1, \dots, f_d on $[n]^d$ and procedures SIFT₀, SIFT₁, \dots , SIFT _{d} taking input in $[n]^d$. SIFT₀ accepts every point in $[n]^d$. For $j \in \{1, \dots, d\}$,

$f_j(x)$ is $f(x)$ if SIFT _{$j-1$} accepts x and is unassigned otherwise.

SIFT _{j} (x) is equal to LINESIFT($x, j; f_j$).

Finally, SIFT is taken to be SIFT _{d} .

The proof that SIFT satisfies [S1-S3] does not rely on the description of the rest of the filter, and the reader may prefer to read those proofs (given in Section 5.5) before continuing with the specification of the function REP.

To specify the function REP, we will need to define several functions that take as argument a d -dimensional box $B = B_1 \times \dots \times B_d$.

The first three functions use the previously defined interval sampling functions SAMPLE₂ ^{j} for $j \in [d]$.

$\text{BOXSAMPLE}(B)$ returns the product set $\prod_{j=1}^d \text{SAMPLE}_2^j(B_j)$.

$\text{SAMPLELINES}(B, j)$ for $j \in [d]$ is \emptyset if $|B_j| < 4$. If $|B_j| \geq 4$, it returns all j -lines L such that $L \cap \text{BOXSAMPLE}(B) \neq \emptyset$. (Equivalently, for $|B_j| \geq 4$, it is the set of lines $\{x^{(j)} : x \in \text{BOXSAMPLE}(B)\}$.)

$\text{SAMPLELINES}(B) = \bigcup_{j=1}^d \text{SAMPLELINES}(B, j)$. Thus, $\text{SAMPLELINES}(B)$ is the set of all lines of $[n]^d$ that pass through a point of $\text{SAMPLE}_2(B)$.

We note a simple yet important fact.

Observation 5.1 *Let P be the j -parent of box B . If L is a j -line in $\text{SAMPLELINES}(B)$, then $L \in \text{SAMPLELINES}(P)$.*

Proof: This is direct consequence of choosing $\text{BOXSAMPLE}(B)$ to be a Cartesian product. The j -line L contains a point x in $\text{BOXSAMPLE}(B)$. The j -line L is specified by values x_r for all $r \neq j$. For $r \neq j$, we have that $B_r = P_r$. This implies $\text{SAMPLE}_2^r(B_r) = \text{SAMPLE}_2^r(P_r)$. By the product structure of BOXSAMPLE , there is a point in $\text{BOXSAMPLE}(P)$ that shares all but its j -coordinate with x . Since L is a j -line it also passes through this point. Hence, $L \in \text{SAMPLELINES}(P)$. \square

We now provide a concise recursive definition of $\text{REFINE}(B)$ using the auxiliary functions $\text{UNSAFELINES}(B)$, $\text{DELETEDLINES}(B)$ which each output a set of lines, and $\text{DELETEDPOINTS}(B)$ which outputs a set of points. Following the recursive definition, we describe a more efficient iterative implementation of $\text{REFINE}(B)$.

$\text{DELETEDLINES}(B)$: This is the union of $\text{UNSAFELINES}(P)$ over all non-left parents P of B .

$\text{DELETEDPOINTS}(B)$: This is the union of all lines in $\text{DELETEDLINES}(B)$.

$\text{REFINE}(B)$ is defined to be $(\text{BOXSAMPLE}(B) \cap \text{Accepted}) - \text{DELETEDPOINTS}(B)$.

$\text{UNSAFELINES}(B)$: This consists of all lines in $L \in \text{SAMPLELINES}(B)$ such that $|B \cap L| \geq 4$ and $\text{REFINE}(B) \cap \text{Upper}(B \cap L)$ is empty. (Recall that for an interval I whose size is a multiple of 4, $\text{Upper}(I)$ consists of the largest $|I|/4$ points of I .)

Observation 5.2 *Let P be a non-left parent of box B and $L \in \text{SAMPLELINES}(B)$. If $L \in \text{UNSAFELINES}(P)$, then $L \in \text{UNSAFELINES}(B)$.*

Proof: Assume $L \in \text{UNSAFELINES}(P)$. Then $L \subseteq \text{DELETEDPOINTS}(B)$ and so $L \cap \text{REFINE}(B) = \emptyset$, and so $L \in \text{UNSAFELINES}(B)$. \square

To evaluate $\text{REFINE}(B)$, we first generate the $(\log n)^{O(d)}$ non-left ancestors of B . Considering each non-left ancestor A in non-decreasing order by size we evaluate $\text{DELETED}(A)$, $\text{REFINE}(A)$ and $\text{UNSAFELINES}(A)$. Note that for a maximal non-left ancestor A of B , $\text{DELETED}(A) = \emptyset$, since A has no non-left parents.

Finally we define $\text{REP}(x)$.

$\text{REP}(x)$: This returns the union of $\text{REFINE}(B)$ over all boxes B that are left-near to x .

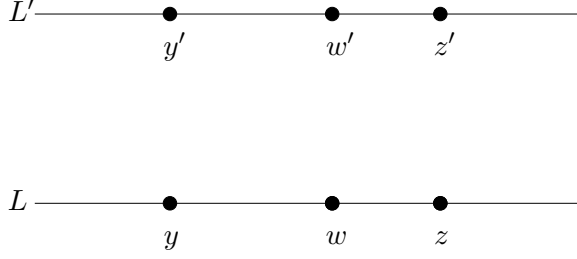


Figure 2: The points y, z, w, y', z', w' in proof that [S1] holds.

5.5 Proof that SIFT satisfies [S1-S3]

This proof is similar to proofs appearing in [1, 3]. 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.

[S1]: We prove by induction that for each $j \in [d]$ and for each $C \in \mathcal{C}_j$, $C \cap Accepted_j$ is f -admissible. The case $j = d$ then gives [S1]. For the base case $j = 1$, \mathcal{C}_1 is the set of lines in direction 1. For $C \in \mathcal{C}_1$, the f -monotonicity of $C \cap Accepted_1$ follows from the fact that $ONEDIMSIFT$ satisfies [S1].

Assume $j \geq 2$ 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 an arbitrary $x \in L$, let $x' \in L'$ be the point such that $x_j = x'_j$. We denote by z the point of L such that $z_j = z'_j$, and by y' the point of L' such that $y'_j = y_j$. See Figure 2. By property [I3], there is an $I \in \mathcal{I}$ such that the corresponding L -interval $I[L]$ is a subset of $[y, z]$ and is near to both y and z . Note that the corresponding interval $I[L']$ contained in line L' is a subset of $[y', z']$ and is near to both y' and z' . Let $S = \text{SAMPLE}_1^j(I)$. When we evaluate $SIFT_j(y)$, y will be rejected if at least half the points of $S[L]$ (the subset of L corresponding to S) are in violation with y with respect to the function f_j . Similarly, when we evaluate $SIFT_j(z')$, z' will be rejected if at least half the points of $S[L']$ (the subset of L' corresponding to S) are in violation with z' with respect to the function f_j .

We now claim that with respect to the function f_j , for each $w \in S[L]$ (the subset of L corresponding to S) either (y, w) is a violation or (w', z') is a violation (see Figure 2). This is clear if either w or w' belongs to $Rejected_{j-1}$ (since then it is unassigned in f_j), so assume that w and w' are both in $Accepted_{j-1}$. We have $w \leq w'$, and w and w' differ only in the first $(j - 1)$ -coordinates (and thus belong to the same class of \mathcal{C}_{j-1}). By the induction hypothesis, $f(w) \leq f(w')$. Since $f(y) > f(z')$ we must have $f(y) > f(w)$ (and therefore (y, w) is a violation) or $f(w') > f(z')$ (and therefore (z', w') is a violation).

It follows that, with respect to f_j , either y is in violation with at least half the points of $S[L]$ or z' is in violation with at least half the points of $S[L']$, which implies that one of them is rejected by $SIFT_j$ and establishes [S1]. □

[S2]: For each $j \in [d]$ we define a failure condition F_j for $SIFT_j$. We show that for each $j \in [d]$ (1) the probability that F_j holds is $|[n]^d|^{-\omega(1)} = n^{-\omega(d)}$, and (2) if none of the conditions F_1, \dots, F_j hold then $|Rejected_j| \leq \frac{1}{9}(10^{j+1} - 10)\varepsilon_f n$.

The failure condition F_j is defined as follows: Along some line in direction j , there is an interval $I \in \mathcal{I}[L]$ and a point $x \in L$ such that (i) I is near to x , (ii) with respect to the partial function f_j , x is in violation with fewer than $.4|I|$ elements of I and (iii) with respect to f_j , x is in violation with at least half the elements of $\text{SAMPLE}_1^j(I)$. For a given j -line L , this is the same failure condition that arose in the analysis of [S2] for the one-dimensional case. We observed that this can happen only if $|I| > t$ and the probability of failure is $O(e^{-\gamma t} n \log n)$ for some constant $\gamma > 0$ (Lemma 4.1). If we take $t = Cd(\log n)^2$ (for a suitable constant C) then this probability is $n^{-C\gamma d \log n}$. If we multiply by the number of lines n^d and the number of pairs (x, I) on each line the result is still $n^{-\omega(d)}$.

So now assume that none of the failure conditions hold. For $j \in [d]$, let ε_j be the distance of f_j from monotonicity. Thus ε_1 is just ε_f .

Claim 5.3 *If none of the failure conditions hold then for each $j \geq 1$,*

$$\mathbf{C1}(j). \quad |\text{Rejected}_j| \leq 10\varepsilon_j n^d$$

$$\mathbf{C2}(j). \quad \varepsilon_{j+1} \leq |\text{Rejected}_j| n^{-d} + \varepsilon_1$$

Given the claim, an easy induction shows that

$$|\text{Rejected}_j| \leq \varepsilon_1 n^d \left(\sum_{i=1}^j 10^i \right) = \varepsilon_f n^d \frac{1}{9} (10^{j+1} - 10),$$

as required for [S2].

Proof: [C1(j)]: Rejected_j is obtained by applying ONEDIMSIFT to each j -line L with respect to the function f_j . By [S2] for ONEDIMSIFT, for any j -line L , we get:

$$|\text{Rejected}_j \cap L| \leq 10\varepsilon_{f_j[L]} n,$$

where $f_j[L]$ is the restriction of f_j to L . Let g_j be a monotone function on $[n]^d$ of distance ε_j from f_j . Summing the previous inequality over all j -lines L , we get:

$$|\text{Rejected}_j| \leq 10 \sum_L \varepsilon_{f_j[L]} n \leq 10n \sum_L d(f_j[L], g_j[L]) = 10n^d d(f_j, g_j) = 10n^d \varepsilon_j,$$

proving [C1(j)].

[C2(j)]: Let g be a monotone function at distance ε_f from f . Since f_{j+1} is obtained from f by unassigning values for $x \in \text{Rejected}_j$, the distance of f_{j+1} from g is at most $n^{-d} |\text{Rejected}_j| + \varepsilon_1$. \square

[S3]: Let T_j be the maximum time needed to evaluate $\text{SIFT}_j(x)$ and let U_j be the maximum time to evaluate $f_j(x)$.

Claim 5.4 *For each $j \geq 1$,*

$$\mathbf{T1}(j). \quad T_j \leq c_1 (\log n)^{c_2} U_j, \text{ for some constants } c_1, c_2.$$

$$\mathbf{T2}(j). \quad U_{j+1} \leq T_j + c_3 \text{ for some constant } c_3.$$

This claim together with a simple induction yields an upper bound on T_j of the form $(\log n)^{O(j)}$, as required.

To see [T1(j)], we observe that $\text{SIFT}_j(x)$ is just ONEDIMSIFT applied to f_j restricted to the 1-line containing x , and so the timing analysis of $\text{SIFT}_j(x)$ applies to get a bound of $c_1((\log n)^{c_2})$ steps where a step may be an evaluation of f_j (which takes time at most U_j).

To see [T2(j)], we note that evaluating $f_{j+1}(x)$ involves evaluating $\text{SIFT}_j(x)$ and an additive constant overhead. \square

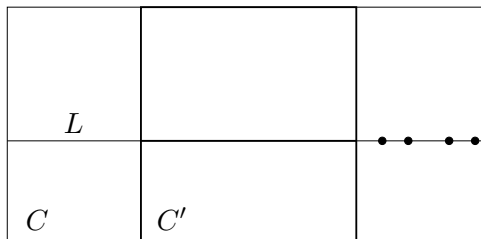


Figure 3: An illustration of REFINED and *BadBoxes*. The box C' is a child of box C . The line L is in $\text{Lines}(C)$. The dark circles indicate the sample points in $\text{Upper}(L \cap C)$. If none of these sample points belongs to $\text{REFINE}(C)$, then L is in $\text{UNSAFE LINES}(C)$, and therefore $\text{REFINE}(C')$ has no point from $L \cap C'$.

5.6 Verification that REP satisfies [B1],[B2],[B3],B5]

[B1] and [B2]: Properties [B1] and [B2], that for all points in $y \in \text{REP}(x)$, $y \leq x$ and $y \in \text{Accepted}$ are immediate from the definition of $\text{REP}(x)$

[B5]: We bound the running time of REP. The number of boxes that are left-near to x is $(\log n)^{O(d)}$. For each such box B we need to compute $\text{REFINE}(B)$. For this, we compute $\text{REFINE}(C)$ and $\text{UNSAFE LINES}(C)$ for each of the at most $(\log n)^{O(d)}$ non-left ancestors C of B (considering them in nonincreasing order of size). For each such C , to compute $\text{REFINE}(C)$ and $\text{UNSAFE LINES}(C)$ we look at each of the at most $(\log n)^{O(d)}$ points $y \in \text{SAMPLE}_2(C)$, 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 $\text{UNSAFE LINES}(C')$ for one of the at most d non-left parents 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 $z \in \text{REP}(x)$. We must show that there is a $z' \in \text{REP}(y)$ 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. Let L be the j -line containing z .

Since $z \in \text{REP}(x)$, there is a box $B = B_1 \times \dots \times B_d$, $B \in \mathcal{B}$ with $z \in \text{REFINE}(B)$, such that B is left-near to x . This implies that for each $h \in [d]$ the interval B_h is left-near to x_h . Let $B = C^1, \dots, C^w$ be the sequence of boxes where for each $i \in [w - 1]$, C^{i+1} is the non-left j -parent of C^i . This is equivalent to the condition that the sequence of (one-dimensional) intervals $B_j = C_j^1, C_j^2, \dots, C_j^w$ is the non-left path from the interval $B_j \in \mathcal{I}$ and for all $i \in [d] - \{j\}$ and $r \in [w]$, $C_i^r = B_i$.

Since $z \in \text{REFINE}(B)$ we have $z \in \text{BOXSAMPLE}(B)$ and so $L \in \text{SAMPLE LINES}(B)$. By repeated application of Observation 5.2, $L \in \text{SAMPLE LINES}(C^i)$ for each $i \in [w]$.

By [I6], we can choose $r \in [w]$ such that the interval C_j^r is left-near to y_j . Also, since C_1 is left-near x_1 , for $i \in [d] - \{j\}$, $C_i^r = C_i$ is left-near to $x_i = y_i$. Therefore the box C^r is left-near to y . If $r = 1$, we have $z \in \text{REFINE}(C^1) \subseteq \text{REP}(y)$ so we can take $z' = z$. So we may assume that $r \geq 2$.

We claim that $\text{REFINE}(C^r) \cap \text{Upper}(C^r \cap L)$ is nonempty. If this is true, then [I5] implies that $\text{Upper}(C^r \cap L)$ is disjoint from (and necessarily to the right of) $B_j \cap L$. Choosing $z' \in \text{REFINE}(C^r) \cap \text{Upper}(C^r \cap L)$ gives a point in $\text{REP}(y)$ that is greater than z .

It remains to show that $\text{REFINE}(C^r) \cap \text{Upper}(C^r \cap L)$ is nonempty. Suppose for contradiction that it is empty. Then $L \in \text{UNSAFE LINES}(C^r)$ (by definition of $\text{UNSAFE LINES}(C^r)$, since $|C^r \cap L| = 2^{r-1}|C^1 \cap L| \geq 2^r \geq 4$). Since $L \in \text{SAMPLE LINES}(C^i)$ for each $i \in [w]$, $(r - 2)$ applications of Observation 5.2 yield $L \in \text{UNSAFE LINES}(C^2) \subseteq \text{DELETED LINES}(C^1)$. Hence $L \subseteq \text{DELETED POINTS}(C^1)$

and so $\text{REFINE}(C^1) \cap L = \emptyset$. This contradicts that $z \in \text{REFINE}(C^1) \cap L$, completing the proof.

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

5.7 Proof that REP satisfies [B4]

Let us define $Excluded = \{x \in [2, n]^d : x \notin \text{REP}(x)\}$. We want an upper bound on $|Excluded|$ as a multiple of $|Rejected|$ that holds with high probability.

Recall that the random seed of the filter consists of strings $s^1, \dots, s^d, r^1, \dots, r^d$. The set $Rejected$ is determined once we fix the strings s^1, \dots, s^d . The analysis of SIFT showed that with high probability the strings are such that we could bound the size of $Rejected$ from above. The analysis in this section holds for any fixed s^1, \dots, s^d , whether or not the resulting set $Rejected$ satisfies the bounds specified by [S2]. Here we are not concerned with the size of $Rejected$, only with the ratio $|Excluded|/|Rejected|$. All probability statements are made with respect to the choice of r^1, \dots, r^d .

We will proceed by constructing a chain of set inclusions that begin with $Excluded$ and end with a set whose size we can readily bound as a multiple of $|Rejected|$. Only one of these inclusions will involve a probabilistic statement, all of the others will hold unconditionally.

We now define the final set in the chain of containments. Recall that for $\theta \in [0, 1]$, a set S is θ -dense in a set I if $|S \cap I| \geq \theta|I|$. Recall also that a subset of $[n]^d$ is a *line interval* if it is an L -interval for some line L . For $S \subseteq [n]^d$:

$\Upsilon_\theta(S)$ denotes the union of all line intervals I (not necessarily in \mathcal{I}) such that S is θ -dense in I .

For $k \geq 1$, $\Upsilon_\theta^k(S)$ is defined inductively by $\Upsilon_\theta^1(S) = \Upsilon_\theta(S)$ and for $k \geq 1$, $\Upsilon_\theta^k S = \Upsilon_\theta(\Upsilon_\theta^{k-1}(S))$.

Lemma 5.5 *For any $S \subseteq [n]^d$ and $k \geq 1$, $|\Upsilon_\theta^k(S)| \leq (\frac{2d}{\theta})^k |S|$.*

Proof: It suffices to prove the result for $k = 1$, since the result for general k follows easily by induction. If L is any line of $[n]^d$ then Lemma 4.2 implies that the union of intervals $I \subseteq L$ such that S is θ -dense in has size at most $\frac{2}{\theta}|L \cap S|$. Summing over all lines in a fixed direction j yields that the union of all j -line intervals I such that S is dense in I has size at most $\frac{2}{\theta}|S|$. Summing over d directions gives that $|\Upsilon_\theta(S)| \leq \frac{2d}{\theta}|S|$. \square

We will show that with high probability (over the choice of the random strings r^1, \dots, r^d):

$$Excluded \subseteq \Upsilon_{2^{-(d+2)}}^d(Rejected) \tag{1}$$

Lemma 5.5 implies $|Excluded| \leq (d2^{d+3})^d |Rejected| = 2^{O(d^2)}$ as required.

Let $A(x)$ be the box $[x_1 - 1, x_1] \times [x_2 - 1, x_2] \times \dots \times [x_d - 1, x_d]$. We will define a suitable function REFINE' mapping each box B to a set $\text{REFINE}'(B) \subseteq B$, and a subset of boxes called *BadBoxes*. Using these, we will build a chain of inclusions that will allow us to prove (1).

$$Excluded \subseteq \bigcup_x (A(x) - \text{REFINE}(A(x))) \subseteq \bigcup_x (A(x) - \text{REFINE}'(A(x))) \subseteq \bigcup_{B \in \text{BadBoxes}} B \subseteq \Upsilon_{2^{-(d+2)}}^d(Rejected)$$

The first part of this chain is easy. The box $A(x)$ is left-maximal for x and so $\text{REFINE}(A(x)) \subseteq \text{REP}(x)$. Thus:

$$Excluded \subseteq \bigcup_x (A(x) - \text{REFINE}(A(x))). \tag{2}$$

5.7.1 The function REFINE'

Next we define for each box B a function $\text{REFINE}'(B)$ which is similar to $\text{REFINE}(B)$ but whose definition does not involve any randomness (except for the randomness used in defining SIFT , which we already fixed). We will show in this section that with high probability, for all $x \in [2, n]^d$, $\text{REFINE}'(A(x)) \subseteq \text{REFINE}(A(x))$. This implies that with high probability the next part of the chain holds:

$$\bigcup_x (A(x) - \text{REFINE}(A(x))) \subseteq \bigcup_x (A(x) - \text{REFINE}'(A(x))). \quad (3)$$

The definition of REFINE involved six functions taking as argument a box B . The first of these was $\text{BOXSAMPLE}(B)$ which returns a small sample product subset of B . The remaining 5 functions all depended on BOXSAMPLE . We now define variants of these 5 functions that are defined by considering all points in B rather than just $\text{BOXSAMPLE}(B)$.

$\text{SAMPLELINES}'(B, j)$ for $j \in [d]$ is the empty set if $|B_j| < 4$. If $|B_j| \geq 4$, it returns all j -lines that have nonempty intersection with B .

$$\text{SAMPLELINES}'(B) = \bigcup_{j=1}^d \text{SAMPLELINES}'(B, j).$$

$\text{DELETED}'(B)$: This is the union of $\text{UNSAFELINES}'(P)$ over all non-left parents P of B .

$\text{REFINE}'(B)$ is obtained from B by removing all points rejected by SIFT and all points that lie in any line of $\text{DELETED}'(B)$.

$\text{UNSAFELINES}'(B)$ is the set of lines $L \in \text{SAMPLELINES}'(B)$ ($|L \cap B| \geq 4$) for which $|\text{REFINE}'(B) \cap \text{Upper}(L \cap B)| < |L \cap B|/8$.

Recall that for $\text{UNSAFELINES}(B)$ is the set of lines $L \in \text{SAMPLELINES}(B)$ for which $\text{REFINE}(B) \cap \text{Upper}(L \cap B)$ is empty, so the definition of $\text{UNSAFELINES}'(B)$ is not directly analogous to the definition of $\text{UNSAFELINES}(B)$. Intuitively, it is “easier” for a line to be in $\text{UNSAFELINES}'(B)$ than in $\text{UNSAFELINES}(B)$, and this will be crucial in proving that $\text{REFINE}'(A(x)) \subseteq \text{REFINE}(A(x))$ with high probability. We now proceed with this proof.

We say that REP *succeeds* if for all boxes B , $\text{UNSAFELINES}(B) \subseteq \text{UNSAFELINES}'(B)$.

Lemma 5.6 *The probability that REP fails is at most $n^{O(d)}2^{-\Omega(t)}$.*

We remind the reader that we consider s^1, \dots, s^d as fixed, and this probability statement is made with respect to the choice of r^1, \dots, r^d .

Proof: Let us say that a box-line pair (B, L) is *relevant* if:

- $|L \cap B| \geq 4$, and
- $|\text{Upper}(L \cap B) \cap \text{REFINE}'(B)| \geq |L \cap B|/8$

Observe that whether (B, L) is relevant is determined by the fixing of s^1, \dots, s^d and is independent of r^1, \dots, r^d .

For a relevant box-line pair (B, L) we define $F(B, L)$ to be the event that:

- $L \in \text{SAMPLELINES}(B)$, and
- $\text{BOXSAMPLE}(B) \cap \text{Upper}(L \cap B) \cap \text{REFINE}'(B) = \emptyset$.

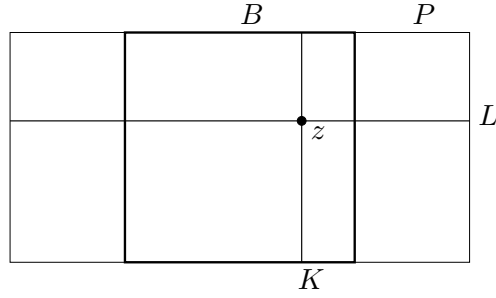


Figure 4: In the proof of Claim 5.7, B is a box, P is the non-left j -parent of B and L is a line in $\text{UNSAFELINES}(B)$. z is an arbitrary point in $\text{BOXSAMPLE}(B) \cap \text{Upper}(L \cap B) \cap \text{Accepted}$. This implies that there is a line $K \in \text{UNSAFELINES}(P)$ through x .

Claim 5.7 *If REP fails then one of the events $F(B, L)$ happens.*

Proof: Suppose that REP fails. Let B be a box of maximum size for which $\text{UNSAFELINES}(B) \not\subseteq \text{UNSAFELINES}'(B)$. Let $L \in \text{UNSAFELINES}(B) - \text{UNSAFELINES}'(B)$. Since $L \in \text{UNSAFELINES}(B)$, we have $|L \cap B| \geq 4$. Since $L \notin \text{UNSAFELINES}'(B)$, $|\text{Upper}(L \cap B) \cap \text{REFINE}'(B)| \geq |L \cap B|/8$. This implies that (B, L) is a relevant pair. We will show that $F(B, L)$ occurs. Since $L \in \text{UNSAFELINES}(B)$, $L \in \text{SAMPLELINES}(B)$ and it suffices to show that the second condition of $F(B, L)$ holds.

Let z be an arbitrary point in $\text{BOXSAMPLE}(B) \cap \text{Upper}(L \cap B)$ as illustrated in Figure 4. It suffices to show that $z \notin \text{REFINE}'(B)$. This is clear if $z \in \text{Rejected}$, so assume $z \in \text{Accepted}$. Since $L \in \text{UNSAFELINES}(B)$, $\text{REFINE}(B) \cap L = \emptyset$, implying $z \notin \text{REFINE}(B)$. But since $z \in \text{BOXSAMPLE}(B) \cap \text{Accepted}$ we must have $z \in \text{DELETEDPOINTS}(B)$, which implies that there is a line K (possibly L) containing z and a non-left parent P of B such that $K \in \text{UNSAFELINES}(P)$, as depicted in Figure 4. By the maximality of B , $K \in \text{UNSAFELINES}'(P)$ which implies $z \in K \subseteq \text{DELETEDPOINTS}'(B)$ and hence $z \notin \text{REFINE}'(B)$, as required to prove the claim. \square

By the claim,

$$\Pr[\text{REP fails}] \leq \Pr\left[\bigcup F(B, L)\right] \leq \sum \Pr[F(B, L)],$$

where the union and sum are over all relevant pairs (B, L) . We now fix a relevant pair (B, L) and bound $\Pr[F(B, L)]$ from above. Let j be the direction of L . Let us further condition on $(r^i : i \in [d] - j)$ (so that the only randomness remaining is r^j). The conditioning determines whether $L \in \text{SAMPLELINES}(B)$. If $L \notin \text{SAMPLELINES}(B)$ then $F(B, L)$ does not occur, so consider a fixing of $(r^i : i \in [d] - j)$ for which $L \in \text{SAMPLELINES}(B)$.

Now $F(B, L)$ happens if and only if $\text{BOXSAMPLE}(B)$ contains no points of $\text{Upper}(L \cap B) \cap \text{REFINE}'(B)$. Given $L \in \text{SAMPLELINES}(B)$, this is equivalent to $\text{SAMPLE}_2^j(B_j)$ contains no points from the set $H_j = \{z_j : z \in \text{Upper}(L \cap B) \cap \text{REFINE}'(B)\}$. Note that $H_j \subseteq \text{Upper}(B_j)$ and by the second condition on relevant pairs, $|H_j| \geq |B_j|/8$. If $t \geq |B_j|$ then $\text{SAMPLE}_2^j(B_j)$ contains all of H_j , so $F(B, L)$ does not occur. If $t < |B_j|$ then $\text{SAMPLE}_2^j(B_j)$ consists of t independent choices from B_j and the probability that they all miss H_j is at most $(7/8)^t$, where t is the length of r^j as specified at the beginning of Section 5.4.

Since the number of relevant pairs is at most $|\mathcal{B}|$ times the number of lines, which is $n^{O(d)}$, the probability that REP fails is at most $n^{O(d)}(7/8)^t$ which is $n^{-\omega(d)}$ for the specified t . \square

Proposition 5.8 *If REP succeeds, then for each point x , $A(x)\text{-REFINE}(A(x)) \subseteq A(x)\text{-REFINE}'(A(x))$.*

Proof: Assume REP succeeds. Then $\text{UNSAFELINES}(B) \subseteq \text{UNSAFELINES}'(B)$ for all boxes $B \in \mathcal{B}$ which implies that $\text{DELETEDPOINTS}(B) \subseteq \text{DELETEDPOINTS}'(B)$ for all boxes $B \in \mathcal{B}$.

Since $\text{BOXSAMPLE}(A(x)) = A(x)$, if $y \in A(x)\text{-REFINE}(A(x))$ then $y \in \text{Rejected} \cup \text{DELETEDPOINTS}(A(x))$. Therefore, $y \in \text{Rejected} \cup \text{DELETEDPOINTS}'(A(x))$, implying $y \in A(x)\text{-REFINE}'(A(x))$. \square

5.7.2 The set *BadBoxes*

For the next inclusion in our chain, we define a set *BadBoxes* of boxes, and show:

$$\bigcup_x (A(x)\text{-REFINE}'(A(x))) \subseteq \bigcup_{B \in \text{BadBoxes}} B. \quad (4)$$

Recall that $\mathcal{B} = \mathcal{B}_d$ was defined to be the set of all boxes $I_1 \times \cdots \times I_d$ where $I_i \in \mathcal{I}$ for each i . We consider an enlarged family of boxes \mathcal{B}^* to be the set of boxes $I_1 \times \cdots \times I_d$ where for each i , $I_i \in \mathcal{I}$, or I_i is a singleton set.

We need some definitions. Let $B = B_1 \times \cdots \times B_d$, $A = A_1 \times \cdots \times A_d$ be boxes in \mathcal{B}^* .

The *type* of B , $\text{Type}(B)$ is the set $\{j \in [d] : |B_j| \geq 2\}$. Let $\mathcal{B}^*(J)$ denote the set of boxes of type J .

Note that there is a natural bijection between $\mathcal{B}^*(J)$ and the set $\mathcal{B}_{|J|}^*$ of boxes $I_1 \times \cdots \times I_{|J|}$ with each $I_i \in \mathcal{I}$. We impose the same digraph structure on $\mathcal{B}^*(J)$ as for $\mathcal{B}_{|J|}^*$, including the notions of parent, child, ancestor, etc.

The *rank* of B , $rk(B)$ is the size of $\text{Type}(B)$. Note that B has rank d if and only if $B \in \mathcal{B}$.

Box A is a *restriction* of B if for all $j \in [d]$ either (1) $A_j = B_j$ or (2) $|A_j| = 1$ and $A_j \subseteq B_j$. (A restriction of B is obtained by shrinking some of the sets B_j for $j \in \text{Type}(B)$ to a single point.) We say that A is a *restriction of B by I* if $I \subseteq [d]$ is the set of indices for which $|A_i| = 1$ and $|B_i| > 1$. It follows that $I = \text{Type}(B) - \text{Type}(A)$.

Observation 5.9 *Let $B, C \in \mathcal{B}$ with $B \subseteq C$ and let A be a restriction of C . If $A \cap B \neq \emptyset$ then $A \cap B$ is a restriction of B . Furthermore, $\text{Type}(A \cap B) = \text{Type}(A)$.*

Proof: For each $j \in \text{Type}(A)$ we have $A_j = C_j$ and $A_j \cap B_j = B_j$. For $j \notin \text{Type}(A)$, A_j is a singleton belonging to B_j (since $A \cap B \neq \emptyset$), in which case $A_j \cap B_j = A_j$. Therefore $A \cap B$ is a restriction of B whose type is $\text{Type}(A)$.

Box A is a *j -slice* of B if it is a restriction of B by the set $\{j\}$. Observe that B has exactly $|B_j|$ j -slices, and they partition B .

BadBoxes is a subset of \mathcal{B}^* defined inductively as follows. Consider boxes in increasing order of rank, and for boxes of the same rank, in decreasing order of size. A box of rank 0, which contains a single point, is in *BadBoxes* if the point belongs to *Rejected*. For a box B of rank at least 1, B is put in *BadBoxes* if either of the following hold:

1. Some box B' that contains B and has the same type as B belongs to *BadBoxes*. (Note that B' is considered before B .)

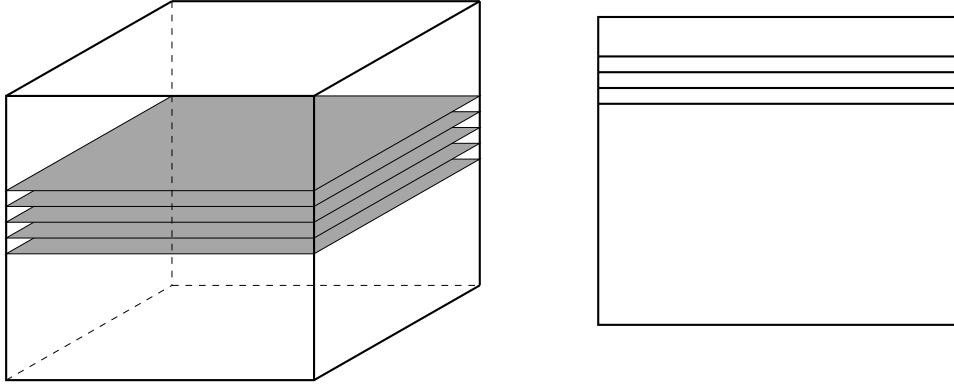


Figure 5: An illustration of *BadBoxes*. For the two-dimensional box on the right, the j -slices are lines. If a sufficiently large fraction of the lines are in *BadBoxes*, then B is in *BadBoxes*. For the three-dimensional box on the left, the j -slices are the gray planes. If a sufficiently large fraction of the gray planes shown are in *BadBoxes*, then B is in *BadBoxes*.

2. For some $j \in \text{Type}(B)$, at least $|B_j|/2^{d+2}$ of the j -slices of B belong to *BadBoxes*. (Note that each such slice has lower rank than B and so is considered before B). (Refer to Figure 5.)

Since $A(x)$ is a full-dimensional box, (4) follows from:

Lemma 5.10 *For any rank d box $B \in \mathcal{B}$, if $x \in B - \text{REFINE}'(B)$ then there is a restriction W of B that contains x such that $W \in \text{BadBoxes}$.*

Proof: We prove by (reverse) induction on the box size. Let B be the largest box (the whole domain). If $x \notin \text{REFINE}'(B)$, then $x \in \text{Rejected}$. We can choose $W = \{x\}$.

Now suppose (by the induction hypothesis) that the lemma holds for all boxes larger than B . Let $x \in B - \text{REFINE}'(B)$. Again, if $x \in \text{Rejected}$, then we can choose $W = \{x\}$.

So assume $x \in \text{Accepted}$. Since $x \in B - \text{REFINE}'(B)$, there is a parent P of B and line $L \in \text{UNSAFELINES}'(P)$ with $x \in P \cap L$. Let j be the direction of L . Recall that this implies that $P_i = B_i$ for all $i \neq j$ and $B_j \subset P_j$. We now give a claim that will immediately imply the lemma.

Claim 5.11 *There is a restriction D of P with $x \in D$ such that $D \in \text{BadBoxes}$.*

Suppose this is true. By Observation 5.9, since $D \cap B$ is non-empty (it contains x), $D \cap B$ is a restriction of B of the same type as D . Since D is in *BadBoxes*, $D \cap B$ is also in *BadBoxes*.

Proof of Claim 5.11. Let $S = L \cap (P - \text{REFINE}'(P))$. Since $L \in \text{UNSAFELINES}'(P)$, $|S| \geq |L \cap P|/8 = |P_j|/8$. In Figure 6, the square points indicate the points of S . By the induction hypothesis (for Lemma 5.10), for each $y \in S$, there is a restriction $D(y)$ of P such that $y \in D(y)$ and $D(y) \in \text{BadBoxes}$. Let $J(y) = \text{Type}(D(y))$.

Case 1: There exists a $y \in S$ such that $j \in J(y)$. Since $D(y)_j = P_j$, and x and y differ only on coordinate j , we have $x \in D(y)$. So $D(y)$ satisfies the claim. (See Figure 6.)

Case 2: For all $y \in S$, $j \notin J(y)$. For each $y \in S$, $J(y)$ is one of the 2^{d-1} nonempty subsets of $[d] - j$. See Figure 7. By the pigeonhole principle, we may choose $S^* \subseteq S$ and $J^* \subseteq [d] - j$ such that $J(y) = J^*$ for all $y \in S^*$ and $|S^*| \geq |S|/2^{d-1} \geq |P_j|/2^{d+2}$.

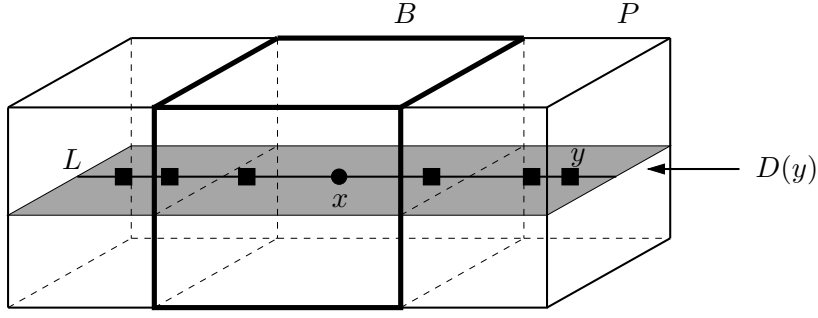


Figure 6: Case 1 for the proof of Claim 5.11. The gray plane is $D(y)$.

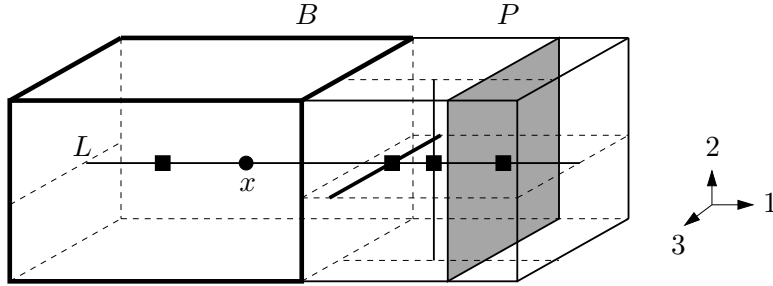


Figure 7: Case 2 for proof of Claim 5.11: (For clarity, the part of P extending beyond B to the left is not shown.) Here the direction j is 1, so $J(y)$ is one of the sets $\{2\}$, $\{3\}$, $\{2, 3\}$ for all $y \in S$. The three possibilities for $D(y)$ are lines (in P) along the 2-direction, the 3-direction, or a slice of P spanning directions 2 and 3. This is depicted in the figure for the three (square) points of S .

Let D be the box defined by $D_i = P_i$ for $i \in J^* \cup \{j\}$ and $D_i = \{x_i\}$ for $i \notin J^* \cup \{j\}$. See Figure 8. Then D is a restriction of P that contains x . Furthermore each of the boxes $D(y)$ for $y \in S^*$ is a j -slice of D (obtained by restricting the j th coordinate to y_j .) Thus D has at least $|S^*| \geq |D_j|/2^{d+2}$ j -slices that belong to $BadBoxes$ and so, by definition, is in $BadBoxes$. This completes the proof of the claim, and the lemma. \square

5.8 The final containment

We complete the chain of containments by proving that for $\theta = 1/2^{d+2}$:

$$\bigcup_{B \in BadBoxes} B \subseteq \Upsilon_\theta^d(Rejected). \quad (5)$$

We need to prove that for any $B \in BadBoxes$, $B \subseteq \Upsilon_\theta^d(S)$.

Suppose the result is false, and let B be a counterexample such that $j = rk(B)$ is minimum, and $|B|$ is maximum among all counterexamples of rank j . If $j = 1$, then $B \in BadBoxes$ implies that at least $1/2^{d+2}$ of the points of B are rejected by SIFT. So $B \subseteq \Upsilon_\theta^1(S)$.

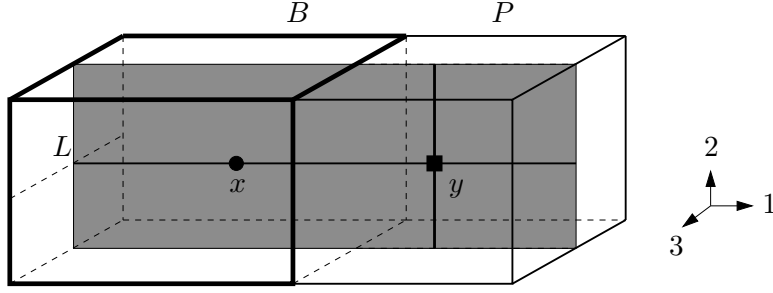


Figure 8: Constructing D . Here $J^* = \{2\}$. For any $y \in S^*$, extending $D(y)$ in direction 1 gives D . The restriction D is the gray plane. If instead $J^* = \{2, 3\}$, then D would be all of P .

We may assume $j \geq 2$. By the maximality of $|B|$, there is no $C \in \text{BadBoxes}$ with $B \subseteq C$ and $\text{rk}(C) = j$. Therefore, by definition of BadBoxes , there is a non-degenerate direction i of B such that at least $1/2^{d+2}$ of the r -slices of B belong to BadBoxes . Each r -slice of B has rank $j - 1$. By the rank minimality of the counterexample B , any such slice that belongs to BadBoxes must be a subset of $\Upsilon_\theta^{j-1}(S)$. Therefore, for each i -line L that meets B , at least a $\theta = 1/2^{d+2}$ fraction of the points of $L \cap C$ belong to $\Upsilon_\theta^{j-1}(S)$. It follows from the definition of Υ_θ^j that $L \cap C \subseteq \Upsilon_\theta^j(S)$ for all L in direction i meeting B , which implies $B \subseteq \Upsilon_\theta^j(S)$.

Combining containments (2), (3) (which holds with high probability), (4) and (5) yields (1), which suffices to establish [B4].

6 Lower bounds

Both the running time and error blow-up of our filter for monotonicity have an exponential dependence on the dimension d . In this section we prove that this is unavoidable. For the sake of proving this lower bound, we will deal with reconstruction on the Boolean hypercube, $H = \{0, 1\}^d$. We will show lower bounds for monotonicity filters for H . Note that the following does not rule out *sublinear* time filters for monotonicity on the hypercube. It only claims that an exponential dependence on d is unavoidable. This does show a complexity gap between testing and reconstruction for the hypercube, since there are monotonicity testers with only a polynomial dependence on d [14, 16, 23].

Theorem 6.1 *Let R be any randomized monotonicity filter for functions on $\{0, 1\}^d$. Then for a suitably small constant $\alpha > 0$ (independent of d), if R answers queries within time $2^{\alpha d}$ then there is an input function f such that R applied to f has error blow-up $2^{\alpha d}$ with probability at least $1/2(1 - 2^{-\alpha d})$.*

Proof: We construct a distribution over functions on $\{0, 1\}^d$ and prove a lower bound for deterministic algorithms over this distribution. Concretely, we prove that for any deterministic filter F with running time bounded by $2^{\alpha d}$ applied to a function chosen from this distribution, F incurs error blow-up at least $2^{\alpha d}$ with probability at least $1/2 - 2^{-\alpha d}$. By Yao's minimax lemma (Chapter 2 of [28]), this implies that for any randomized filter R with time bound $2^{\alpha d}$, there is an input function f such that when R is run on f , it has error blow-up $2^{\alpha d}$ with probability at least $1/2(1 - 2^{-\alpha d})$.

For $x \in \{0, 1\}^d$, the *weight* of x , $w(x)$, is $\sum_i x_i$. For convenience, assume d is a multiple of 8. For a point q of weight $d/2$, define the function h_q to be 0 on the set $Z(q) = \{z : (z \leq q) \wedge (w(z) \geq d/8)\}$

and 1 elsewhere. Define the function g_q to be 0 on the set $\{z : (z \leq q)\}$ and 1 elsewhere. Note that g_q is monotone. Let $S(q) = \{z : (z \leq q) \wedge (w(z) < d/8)\}$ be the set of places where f_q and g_q differ. Observe that

$$\frac{|Z(q)|}{|S(q)|} = \frac{\sum_{j=d/8}^{d/2} \binom{d/2}{j}}{\sum_{j=0}^{d/8-1} \binom{d/2}{j}} \geq 2^{\alpha d},$$

for some suitable $\alpha > 0$.

The function f is chosen as follows. With probability $1/2$, f is the constant 1 function. With probability $1/2$, we choose a point q uniformly at random from points of weight $d/2$ and take $f = h_q$.

Fix a deterministic query algorithm which, on input a point $x \in \{0, 1\}^d$, queries f at at most $t = 2^{\alpha d}$ places, and then outputs $g(x)$, the corrected value of f . The algorithm adaptively queries the function f at points, where each point to be queried may depend on the answer to previous queries.

Consider the behavior of the algorithm in the case that the input is $x = 0^d$ and the function f is the constant 1 function. Suppose the (deterministic) sequence of queries is x_1, \dots, x_t . For *any* input algorithm f with $f(x_i) = 1, \forall i \in [t]$, the sequence of queries will be x_1, \dots, x_t . Furthermore, the output will be the same.

Suppose in this scenario the algorithm decided $g(0^d) \neq 1$. If the input was the monotone constant 1 function, then the error blow-up is infinite. Since f is the constant 1 function with probability $1/2$, with probability $1/2$ the algorithm incurs infinite blow-up. We can therefore assume that the algorithm outputs $g(0^d) = 1$ when f is the constant 1 function.

Now with probability $1/2$, f is chosen to be h_q for some q chosen uniformly at random. Let us first note that the probability (with respect to q) that one of the points x_1, \dots, x_t belongs to $Z(q)$ is small. Indeed, for fixed y of weight j , this probability is 0 if $j \notin [d/8, d/2]$ and is $\binom{d/2}{j} / \binom{d}{j}$ for $j \in [d/8, d/2]$, which is bounded above by $2^{-2\alpha d}$ for a suitable $\alpha > 0$. Applying a union bound over queries we have that the probability that at least one of the points x_1, \dots, x_t belongs to $Z(q)$ is at most $2^{-\alpha d}$.

Thus, with probability $1/2(1 - 2^{-\alpha d})$, the function f is equal to h_q for some q , but the set x_1, \dots, x_t is disjoint from $Z(q)$. On input point 0^d , the algorithm will behave exactly as if f were the constant 1 function, and output $g(0^d) = 1$. In this case it is required, by monotonicity, to change the value of the function on all points in $Z(q)$. Since the function g_q is a monotone function, this means that the error blow-up will be at least $|Z(q)|/|S(q)|$ which, as noted above, is at least $2^{\alpha d}$.

When f is chosen from the distribution, the error blow-up of any deterministic algorithm with time bound $2^{\alpha d}$ is at least $2^{\alpha d}$ with probability at least $1/2(1 - 2^{-\alpha d})$. An application of Yao's minimax lemma completes the proof. \square

7 Further Work

Our monotonicity filter has error blow-up $2^{O(d^2)}$ as compared with the $2^{\Omega(d)}$ lower bound proved in the previous section, so a gap still remains. The online filter constructed in [2] achieves $2^{O(d)}$ error blow-up.

Having introduced the new model of local reconstruction, it is natural to ask whether filters can be designed for other properties. A direct extension on this work would be to study monotonicity filters for other partial orders. There have been some results on property testing for general posets [11, 19]. Monotonicity on the hypercube has been well studied, and designing filters may be the next step in understanding this property on the hypercube. It might be interesting to search for filters with additive error blow-up, instead of multiplicative as discussed.

Filters for graph properties is another possible direction of research. There have been some results for the specific property of expansion for sparse graphs [26]. There are very broad and general theorems about testable properties for graphs [4, 5, 10, 20]. It would be very interesting to see if these theorems can be generalized for reconstruction as well.

8 Acknowledgements

We thank Oded Goldreich and two anonymous referees for pointing out relevant related work and making many suggestions to improve our presentation.

References

- [1] N. Ailon and B. Chazelle. Information theory in property testing and monotonicity testing in higher dimension. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 434–447, 2005.
- [2] 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.
- [3] 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.
- [4] N. Alon, E. Fischer, I. Newman, and A. Shapira. A combinatorial characterization of the testable graph properties : it’s all about regularity. In *Proceedings of the 38th Annual Symposium on Theory of Computing (STOC)*, pages 251–260, 2006.
- [5] N. Alon and A. Shapira. A characterization of the (natural) graph properties testable with one-sided error. In *Proceedings of the 46th Foundations of Computer Science (FOCS)*, pages 429–438, 2005.
- [6] S. Arora and M. Sudan. Improved low-degree testing and its applications. In *Proceedings of the 29th Annual Symposium on Theory of Computing (STOC)*, pages 485–495, 1997.
- [7] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual Symposium on Theory of Computing (STOC)*, pages 21–31, 1991.
- [8] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. PP has subexponential time simulations unless EXP-TIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.
- [9] T. Batu, R. Rubinfeld, and P. White. Fast approximate PCPs for multidimensional bin-packing problems. *Information and Computation*, 196(1):42–56, 2005.
- [10] I. Benjamini, O. Schramm, and A. Shapira. Every minor-closed property of sparse graphs is testable. In *Proceedings of the 40th Annual Symposium on Theory of Computing (STOC)*, pages 393–402, 2008.
- [11] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. Woodruff. Transitive-closure spanners. In *Proceedings of the 18th Annual Symposium on Discrete Algorithms (SODA)*, pages 531–540, 2009.

- [12] 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.
- [13] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45:965–981, 1998.
- [14] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. *Proceedings of the 3rd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 97–108, 1999.
- [15] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Journal of Computer Systems and Sciences (JCSS)*, 60(3):717–751, 2000.
- [16] S. Fattal and D. Ron. Approximating the distance to monotonicity in high dimensions. In <http://www.eng.tau.ac.il/danar/Public-pdf/app-mon-long.pdf>.
- [17] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of EATCS*, 75:97–126, 2001.
- [18] E. Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004.
- [19] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual Symposium on Theory of Computing (STOC)*, pages 474–483, 2002.
- [20] E. Fischer and I. Newman. Testing versus estimation of graph properties. In *Proceedings of the 37th Annual Symposium on Theory of Computing (STOC)*, pages 138–146, 2005.
- [21] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the 23rd Annual Symposium on Theory of Computing (STOC)*, pages 32–42, 1991.
- [22] O. Goldreich. Combinatorial property testing - a survey. *Randomization Methods in Algorithm Design*, pages 45–60, 1998.
- [23] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20:301–337, 2000.
- [24] 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.
- [25] S. Halevy and E. Kushilevitz. Testing monotonicity over graph products. *Random Structures and Algorithms*, 33(1):44–67, 2008.
- [26] S. Kale, Y. Peres, and C. Seshadhri. Noise tolerance of expanders and sublinear expander reconstruction. In *Proceedings of the 49th Foundations of Computer Science (FOCS)*, pages 719–728, 2008.
- [27] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the 32th Annual Symposium on Theory of Computing (STOC)*, pages 80–86, 2000.

- [28] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [29] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 6(72):1012–1042, 2006.
- [30] D. Ron. Property testing. *Handbook on Randomization*, II:597–649, 2001.
- [31] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal of Computing*, 25:647–668, 1996.
- [32] M. Saks and C. Seshadhri. Parallel monotonicity reconstruction. In *Proceedings of 19th Annual Symposium on Discrete Algorithms (SODA)*, pages 962–971, 2006.
- [33] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.