Estimating the Distance to a Monotone Function*

Nir Ailon† Bernard Chazelle† Seshadhri Comandur† Ding Liu†

October 3, 2005

Abstract

In standard property testing, the task is to distinguish between objects that have a property $\mathcal{P}$ and those that are ε-far from $\mathcal{P}$, for some $\varepsilon > 0$. In this setting, it is perfectly acceptable for the tester to provide a negative answer for every input object that does not satisfy $\mathcal{P}$. This implies that property testing in and of itself cannot be expected to yield any information whatsoever about the distance from the object to the property. We address this problem in this paper, restricting our attention to monotonicity testing. A function $f : \{1, \ldots, n\} \mapsto \mathbf{R}$ is at distance ε_f from being monotone if it can (and must) be modified at $\varepsilon_f n$ places to become monotone. For any fixed $\delta > 0$, we compute, with probability at least $2/3$, an interval $[(1/2-\delta)\varepsilon, \varepsilon]$ that encloses ε_f. The running time of our algorithm is $O(\varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$, which is optimal within a factor of $\log\log \varepsilon_f^{-1}$ and represents a substantial improvement over previous work. We give a second algorithm with an expected running time of $O(\varepsilon_f^{-1} \log n \log\log\log n)$. Finally, we extend our results to multivariate functions.

1 Introduction

Since the emergence of property testing in the nineties [9,15], great progress has been made on a long list of combinatorial, algebraic, and geometric testing problems; see [5,7,14] for surveys. Property testing is a relaxation of the standard decision problem: Given a property $\mathcal{P}$, instead of determining exactly whether a given input object satisfies $\mathcal{P}$ or not, we only need to

*This work was supported in part by NSF grants CCR-998817, CCR-0306283, ARO Grant DAAH04-96-1-0181.

†Dept. Comp. Sci., Princeton University, { nailon, chazelle, csesha, dingliu }@cs.princeton.edu

differentiate between the cases where the object satisfies the property and it is far from doing so.

This involves a notion of distance: Typically the object is said to be ε-far from $\mathcal{P}$ if at least a fraction ε of its description must be modified in order to enforce the property. The smallest such ε is called the distance of the object to $\mathcal{P}$. In this setting, the tester can say "no" for every input object that does not satisfy $\mathcal{P}$, which precludes the leaking of any information regarding the distance of the object to the property.

This weakness has led Parnas, Ron, and Rubinfeld [13] to introduce the concept of *tolerant* property testing. Given $0 \leq \varepsilon_1 < \varepsilon_2 \leq 1$, a tolerant tester must accept all inputs that are not ε_1-far from $\mathcal{P}$ and reject all of those that are ε_2-far (and output anything it pleases otherwise). A related problem studied in [13] is that of estimating the actual distance of the object to the property within prescribed error bounds. A tolerant tester can be constructed (this construction will be described later) by using an algorithm that estimates distance. In the model considered, all algorithms are randomized and err with probability at most 1/3 (or equivalently any arbitrarily small constant).

Testing the monotonicity of functions has been extensively studied [2–4, 6, 8, 10]. In the one-dimensional case, given a function $f : \{1, \ldots, n\} \mapsto \mathbf{R}$, after querying $O(\log n)/\varepsilon$ function values, we can, with probability at least 2/3, accept f if it is monotone and reject it if it is ε-far from being monotone [4]. These methods do not provide for tolerant property testing, however. Very recently, Parnas, Ron and Rubinfeld [13] designed sublinear algorithms for tolerant property testing and distance approximation for two problems: function monotonicity and clustering. If ε_f denotes the distance of f to monotonicity, their algorithm computes an estimate $\hat{\varepsilon}$ for ε_f that satisfies $(1/2)\varepsilon_f - \delta \leq \hat{\varepsilon} \leq \varepsilon_f + \delta$ with high probability. The query complexity and running time of their algorithm are both $\tilde{O}((\log n)^7/\delta^4)$ (the $\tilde{O}$ notation hides a factor of $(\log \log n)^{O(1)}$). The algorithm maintains and queries a data structure called an "index-value tree." Since the running time is sublinear, the tree is stored implicitly and only relevant portions are constructed whenever necessary, using random sampling to make approximate queries on the tree. Their construction is sophisticated and highly ingenious, but all in all quite involved.

We propose a simpler, faster, algorithm that is nearly optimal. Given any fixed $\delta > 0$, it outputs an interval $[(1/2 - \delta)\varepsilon, \varepsilon]$ that encloses ε_f with probability at least 2/3. The running time is $O(\varepsilon_f^{-1} \log \log \varepsilon_f^{-1} \log n)$, which is optimal within a factor of $\log \log \varepsilon_f^{-1}$. (A sketch of the optimality proof

is provided later on.) One thing to note is the different use of δ: in our algorithm it is part of the multiplicative factor, whereas in [13] it is an additive term. To achieve the same multiplicative factor as in our algorithm, the additive term needs to be $\Theta(\delta\varepsilon_f)$. This makes the running time of Parnas et al.'s algorithm $\tilde{O}((\log n)^7/\varepsilon_f^4)$, for any fixed δ.

The starting point of our algorithm is the property tester of Ergun et al. [4], which relies on a key fact: There are at least $\varepsilon_f n$ "critical" integers $i \in \{1, \ldots, n\}$; for i to be critical means that it is the (left or right) endpoint of an interval at least half of whose elements are in violation with i. Here i is said to violate j if either $i < j$ and $f(i) > f(j)$ or $i > j$ and $f(i) < f(j)$. By proving an upper bound on the number of critical integers, we are able to define a "signature" distribution for f which reflects its distance ε_f fairly accurately. Specifically, two functions with distances to monotonicity off by a factor of 2 (roughly) will have signatures that are distinguishable in time $O(\varepsilon_f^{-1} \log n)$. This provides us with a tolerant property tester for monotonicity. We can turn it into a distance approximator by using a one-way searching strategy, which we discuss below.

Just as in [13], our algorithm extends to higher dimension We denote the dimension by d, and consider functions $f : \{1, \ldots, n\}^d \mapsto \mathbf{R}$. The poset considered is the d-fold product poset [11] of $\{1, \ldots, n\}$. In other words, for $\mathbf{x}, \mathbf{y} \in \{1, \ldots, n\}^d$, $\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$ for all $i = 1, \ldots, d$. The distance of f from monotonicity is ε_f if it can (and must) be changed at $\varepsilon_f n^d$ places to make it monotone.

Theorem 1.1 *There exists an algorithm that, given a function $f : \{1, \ldots, n\}^d \mapsto \mathbf{R}$, computes a value ε in time $O(2^d d \varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$ such that $\varepsilon_f \in [(1/2 - \delta)d^{-1}2^{-d-1}\varepsilon, \varepsilon]$ with probability at least $2/3$.*

We also present an improvement of our one-dimensional algorithm for small enough values of ε. We show how to estimate ε_f in time $O(\varepsilon_f^{-1} \log n \log\log\log n)$. Unlike in our previous algorithm, the number of steps in this one is itself a random variable; therefore, the running time is to be understood in the expected sense over the random bits used by the algorithm.

2 Preliminaries

For ease of notation, we let $[n] = \{1, \ldots, n\}$. Given two functions $f, g : [n] \mapsto \mathbf{R}$, let $d(f, g) = \text{Prob}[f(x) \neq g(x)]$ denote the distance between f and g, where $x \in [n]$ is chosen uniformly at random.

Now fix a function $f : [n] \mapsto \mathbf{R}$. We define the distance ε_f of f to monotonicity as $\min_{g \in \mathcal{M}} d(f, g)$, where $\mathcal{M}$ is the set of monotone functions from $[n]$ to $\mathbf{R}$. Let $C \subseteq [n]$ be a subset of $\varepsilon_f n$ integers over which f can be modified to become monotone. Note that this subset is *not* necessarily uniquely determined by f.

Definition 2.1 *Given $0 < \delta < 1/2$, the integer $i \in [n]$ is called* δ-big *if there exists $j > i$ such that*

$$\left| \left\{ i \leq k \leq j \,|\, f(k) < f(i) \right\} \right| \geq (1/2 - \delta)(j - i + 1)$$

or, similarly, $j < i$ such that

$$\left| \left\{ j \leq k \leq i \,|\, f(k) > f(i) \right\} \right| \geq (1/2 - \delta)(i - j + 1)$$

Intuitively, integer i is big if $f(i)$ violates monotonicity with an abundance of witnesses.

Definition 2.2 *An integer $i \in C$ is called* high-critical *(resp.* low-critical*) if there is $j \notin C$ such that $j > i$ and $f(j) < f(i)$ (resp. $j < i$ and $f(j) > f(i)$).*

Note that no $i \in C$ can be both high-critical and low-critical, because that would imply the existence of $j, j' \notin C$ such that $j < i < j'$ and $f(j) > f(i) > f(j')$, and therefore at least one of j, j' must be in C, a contradiction.

In the following we show that when δ is small, the number of δ-big integers approximates $\varepsilon_f n$ to within a factor of roughly 2.

Lemma 2.3 *For any function $f : [n] \to \mathbb{R}$ -*

(i) At least $\varepsilon_f n$ integers are 0-big; (ii) no more than $(2+4\delta/(1-2\delta))\varepsilon_f n$ integers are δ-big.

Proof: Note that, for any $i < j$ such that $f(i) > f(j)$, either i or j (or both) is 0-big. Therefore, if we were to remove all the 0-big integers from the domain $\{1, \ldots, n\}$, the function f would become monotone; hence (i).

To prove (ii), we start by assigning to each δ-big i a witness j_i to its bigness (if many witnesses exist, we just choose any one). If $j_i > i$, then i is called *right-big*; else it is *left-big*. (Obviously, the classification depends on the choice of witnesses.) For clarity, refer to Figure 1. The bold portions of the function represent are the function values of C. b is low-critical and c is high critical. a is right-big, with b as its witness. Similarly, d is left-big, with c as its witness.

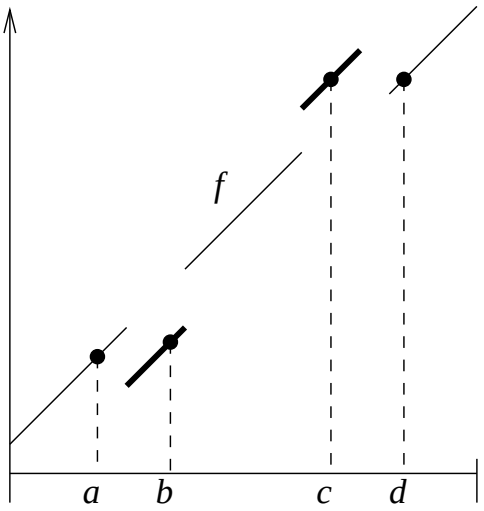

Figure 1: *Definitions*

To bound the number of right-bigs, we charge low-criticals with a credit scheme. (Then we apply a similar procedure to bound the number of left-bigs.) Initially, each element of C is assigned one unit of credit. For each right-big $i \notin C$ among $n, \ldots, 1$ in this order, *spread* one credit among all the low-criticals k such that $i \leq k \leq j_i$ and $f(k) < f(i)$. We use the word "spread" because we do not simply drop one unit of credit into one account. Rather, viewing the accounts as buckets and credit as water, we pour one unit of water one infinitesimal drop at a time, always pouring the next drop into the least filled bucket. More precisely, we repeat the following step until we are left with no more credit to spread: If we currently have z units of credit, the least filled buckets have an amount of μ units of credit each, there are t such buckets, and the second-least filled buckets have an amount of $\mu' > \mu$ credit ($\mu' = \infty$ if there are no second-least filled buckets), then we add to the least-filled buckets $\min\{z/t, \mu' - \mu\}$ units of credit each.

We now show that no low-critical ever receives an excess of $2+4\delta/(1-2\delta)$ units of credit. Suppose by contradiction that this were the case. Let i be the right-big that causes the low-critical k's account to reach over $2+4\delta/(1-2\delta)$. By construction i is not low-critical; therefore, the excess occurs while right-big i is charging the l low-criticals k such that $i < k \leq j_i$ and $f(k) < f(i)$. Note that, because $i \notin C$, any k satisfying these two conditions is

a low-critical and thus gets charged. With the uniform charging scheme, this ensures that all of these l low-criticals have the same amount of credit by the time they reach the excess value, which gives a total greater than $l(2 + 4\delta/(1 - 2\delta))$. By definition of right-bigness, $l \geq (1/2 - \delta)(j_i - i + 1)$. But none of these accounts could be charged before step j_i; therefore,

$$(1/2 - \delta)(j_i - i + 1)(2 + 4\delta/(1 - 2\delta)) < j_i - i + 1,$$

which is a contradiction.

We handle left-bigs in a similar way by running now from left to right, ie, $i = 1, \ldots, n$. Since no integer can be both low-critical and high-critical, part (ii) of the lemma follows. □

3 Estimating Distance to Monotonicity

In this section, we will describe the algorithm that estimates the distance of function to monotonicity. Our aim is to prove the following theorem.

Theorem 3.1 *For any fixed $\delta > 0$, there exists an algorithm* DistApprox that given a function f, calculates a value ε in $O(\varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$ time, such that $\varepsilon_f \in [(1/2 - \delta)\varepsilon, \varepsilon]$ with probability at least 2/3.

We will start by first solving the problem of *distance separation*. The next two subsections will be devoted to proving the following -

Lemma 3.2 (Existence of a *distance separation* algorithm) *For any $\varepsilon > 0$ and fixed (arbitrarily small) constant $\delta > 0$, there exists an algorithm* DistSeparation that decides, in time $O(\varepsilon^{-1} \log n)$ whether $\varepsilon_f > \varepsilon$ or $\varepsilon_f < (1/2 - \delta)\varepsilon$ and with probability at least 2/3. (If $(1/2 - \delta)\varepsilon \leq \varepsilon_f \leq \varepsilon$, the algorithm can report anything.)

3.1 A Separation Oracle

The key to estimating the distance to monotonicity is to approximate the number of the δ-big integers. To identify a δ-big integer i, we need to find an interval starting or ending at i such that there are many violations with i in the interval. This is done through random sampling, to ensure a sublinear running time.

Let $\mathcal{D}$ be the joint distribution of m independent 0/1 random variables $x_1, \ldots, x_m$, which can be sampled independently. If $\mathbf{E}[x_i] \leq a$ for all i, then $\mathcal{D}$ is called *a-light*; else it is *a-heavy*. We describe an algorithm LightTest

```
LightTest1(D, a, b, k)

 For each x ∈ D
   Sample it k times and compute the average x̂
 Form D' = { x ∈ D | x̂ > (a + b)/2 }
 If |D'| = 0 then
   Output "a-light"
 If |D'| ≥ |D|/2 then
   output "b-heavy"
 LightTest1(D', k + c0)
```

Figure 2: LightTest$_1$

which, given any $a < b$, determines whether a distribution is a-light or b-heavy. More precisely -

$$\Pr[\text{LightTest outputs } \textit{"a-light"}] \begin{cases} \geq 2/3 & \text{if } \mathcal{D} \text{ is } a\text{-light} \\ \leq 1/3 & \text{if } \mathcal{D} \text{ is } b\text{-heavy} \\ \text{not guaranteed} & \text{otherwise} \end{cases}$$

We will assume that LightTest$(\mathcal{D}, a, b)$ = LightTest$_1(\mathcal{D}, a, b, c_0)$, where c_0 will be chosen below. The procedure LightTest$_1$ is described in Figure 2.

Lemma 3.3 *If $\mathcal{D}$ is either a-light or b-heavy, for some fixed $a < b$, then with probability* 2/3 LightTest tells which is the case in $O(bm/(b-a)^2)$ time.

Proof: Choose c_0 such that $c_1 \stackrel{\text{def}}{=} c_0(b-a)^2/b$ is a large enough constant. LightTest runs in time proportional to $\sum_{r \geq 0} c_0 r(m/2^r) = O(c_0 m)$. To see why it works, we begin with a simple observation. Suppose that $\mathbf{E}[x_i] > b$, then at the r-th recursive call we sample x_i (if at all) exactly $c_0 r$ times; therefore, by Chernoff's bounds,

$$\text{Prob}[\hat{x}_i \leq (a+b)/2] = 2^{-\Omega(c_1 r)}$$

The same upper bound holds for the probability that $\hat{x}_i > (a+b)/2$, assuming that $\mathbf{E}[x_i] \leq a$. Suppose now that :

- $\mathcal{D}$ IS b-HEAVY: Let x_i be such that $\mathbf{E}[x_i] > b$. At the r-th recursion call, the probability that $\mathcal{D}'$ is empty is $2^{-\Omega(c_1 r)}$. Summing up over all r bounds the likelihood of erring by 1/3.

- $\mathcal{D}$ IS a-LIGHT: The probability that any given $\hat{x}_i$ exceeds $(a+b)/2$ is at most 1/3 (conservatively) and so erring any time before the size of S is recursively reduced to below c_1 is $\sum_{c_1 \leq r < |S|} 2^{-\Omega(r)} = 2^{-\Omega(c_1)} < 1/6$. After that stage, the probability of reaching a b-heavy verdict is at most $O(c_1(\log c_1)2^{-\Omega(c_1)}) < 1/6$. □

3.2 Distance Separation: The Algorithm

We need one piece of terminology before describing the distance separation algorithm, called DISTSEPARATION(refer to Figure 3). Given an interval in $[u, v]$, we define two 0/1 random variables $\alpha[u, v]$ and $\beta[u, v]$: given random $i \in [u, v]$, $\alpha[u, v] = 1$ iff $f(u) > f(i)$ ($\beta[u, v] = 1$ iff $f(v) < f(i)$). We prove below that, with probability at least 2/3, DISTSEPARATION (f, ε, δ) reports that $\varepsilon_f > \varepsilon$ (resp. $\varepsilon_f < (1/2 - \delta)\varepsilon$) if it is, indeed the case, and outputs an arbitrary decision if $(1/2 - \delta)\varepsilon \leq \varepsilon_f \leq \varepsilon$.

The algorithm assumes that both δ and ε/δ are suitably small. The value of δ can naturally be chosen to be sufficiently small. To make ε small, however, we use an artifice: using a slightly abusive notation, we assume that the range of f is $[1, n/\delta]$ and that $f(i) = +\infty$ for $i = n+1, \ldots, O(n/\delta)$. We also need to assume that LIGHTTEST succeeds with probability at least $1 - \delta^2$ (instead of 2/3); to do that it iterates the call to LIGHTTEST$_1$ $O(\log \delta^{-1})$ times and take a majority vote.

Proof: **[Lemma 3.2]** To prove the correctness of the algorithm, it suffices to show that:

- If $\varepsilon_f > \varepsilon$, then $\mathcal{D}$ is $(1/2 - \delta/4)$-heavy with probability $1/2 + \Omega(\delta)$:

  By Lemma 2.3 (i), more than εn integers are 0-big, so the probability of hitting at least one of them in the first step (and hence, of ensuring that $\mathcal{D}$ is $(1/2)/(1 + \delta/4)$-heavy) is at least $1 - (1 - \varepsilon)^s > 1/2 + \Omega(\delta)$.

- If $\varepsilon_f < (1/2 - \delta)\varepsilon$, then $\mathcal{D}$ is $(1/2 - \delta/3)$-light with probability $1/2 + \Omega(\delta)$:

DistSeparation (f, ε, δ)

Pick $s = \lceil (1+\delta/2)\varepsilon^{-1} \ln 2 \rceil$ random $i \in [n]$
For each $1 \le k \le (5/\delta) \ln n$
  Let $x^{(i)}_{2k-1} = \alpha[i, i + (1+\delta/4)^k]$
  $x^{(i)}_{2k} = \beta[i - (1+\delta/4)^k, i]$
Let $\mathcal{D}$ be the distribution of $(x^{(1)}_1, x^{(1)}_2, \ldots, x^{(2)}_1, x^{(2)}_2, \ldots, x^{(s)}_1, x^{(s)}_2, \ldots)$
Let $a = (1/2 - \delta/3), b = (1/2 - \delta/4)$
If LightTest$(\mathcal{D}, a, b)$ = "b-heavy" then
  output "$\varepsilon_f > \varepsilon$"
Else
  output "$\varepsilon_f < (1/2 - \delta)\varepsilon$"

Figure 3: DistSeparation

By Lemma 2.3 (ii), the number of $\delta/3$-big integers is less than $(1-\delta)\varepsilon n$; therefore, the probability of missing all of them (and hence, of ensuring that $\mathcal{D}$ is $(1/2 - \delta/3)$-light) is at least $(1 - (1-\delta)\varepsilon)^s > 1/2 + \Omega(\delta)$.

By running DistSeparation $O(1/\delta^2)$ times and taking a majority vote, we can boost the probability of success to 2/3. By Lemma 3.3 (and noting that $m = sk = O(\varepsilon^{-1} \log n)$), we conclude that the running of DistSeparation (including the majority vote boosting) is $O(\varepsilon^{-1} \log n)$. □

3.3 From Separation to Estimation

It is easy to reduce the problem of estimating the distance to a "distance separation" decision problem. We have an algorithm DistSeparation(f, ε, δ) running in time $O(\varepsilon^{-1} \log n)$ and returning either "$\varepsilon_f > \varepsilon$" or "$\varepsilon_f < (1/2 - \delta)\varepsilon$" with the following guarantee:

$$\Pr[\text{DistSeparation outputs "}\varepsilon_f > \varepsilon\text{"}] \begin{cases} \ge 2/3 & \text{if } \varepsilon_f > \varepsilon \\ \le 1/3 & \text{if } \varepsilon_f < (1/2 - \delta)\varepsilon \\ \text{not guaranteed} & \text{otherwise} \end{cases}$$

Consider the following algorithm DistApprox$_1$ in Figure 4. The constant c_1 in the algorithm is determined below.

```
DistApprox_1(f)

For k = 1, 2, ...
    Let ε_k = (1/2 − δ)^k
    Run DistSeparation(f, ε_k, δ) c_1 log(k + 1) times
    Let M be a majority vote on the outputs
    If M = "ε_f > ε_k"
        Return k
```

Figure 4: DistApprox$_1$

Fix an input f to DistApprox$_1$. Let ℓ be the random variable returned by DistApprox$_1(f)$. If c_1 is adequately chosen, then by Chernoff's bound, the probability that $\varepsilon_{\ell+1} \leq \varepsilon_f \leq \varepsilon_{\ell-1}$ is at least $1 - \sum_{k\geq 0} O(1/k^2) > 5/6$. In this case, the running time is $\sum_{1\leq k\leq \ell} O(\log(k+1))\varepsilon_k^{-1} \log n = O(\varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$, as claimed. Moreover, it is also not hard to see that for c_1 chosen above, for any $j \geq 2$, the probability that the algorithm returns ε_ℓ such that $\varepsilon_{\ell-j} \leq \varepsilon_f \leq \varepsilon_{\ell-j-1}$ is $e^{-\Omega(c_1 j \log j)}$, giving a tight concentration bound for the claimed running time.

Algorithm DistApprox$_1$ does not quite do the job. Indeed, we are now left with the knowledge that $\varepsilon_f \in [\varepsilon_{\ell+1}, \varepsilon_{\ell-1}]$. Since we are trying to find an interval of the form $[(1/2-\delta)\varepsilon, \varepsilon]$, the present interval is not good enough. We do know that $\varepsilon_f \in [\varepsilon_{\ell-1}/5, \varepsilon_{\ell-1}]$. To get the desired interval, we first run DistApprox$_1(f)$ to obtain ℓ, and then run another procedure DistApprox$_2(f, \ell)$. The procedure DistApprox$_2$ described in Figure 5 is very similar to DistApprox$_1$, except that the sequence $\{\varepsilon_k\}$ is an arithmetic sequence, and not a geometric one. The constant $c_2 > 0$ will be chosen below.

Using similar analysis as that of DistApprox$_1$, we conclude that by choosing c_2 large enough we can now pinpoint ε_f to within an interval of the form $[(1/2-O(\delta))\varepsilon, \varepsilon]$ with probability at least 5/6. Therefore, the algorithm DistApprox(f) will simply output DistApprox$_2(f$,DistApprox$_1(f))$. Running DistApprox(f) gives us (up to rescaling of δ) the desired result claimed in Theorem 3.1. Note that the dependence of the running time

```
DistApprox_2(f, ℓ)

Let ε_0 = (1/2 − δ)^{ℓ−1}
/* Assumption:  ε_f ∈ [ε_0/5, ε_0] */

For k = 1, 2, ..., ⌈4ε_0/(5δ)⌉
    Let ε_k = (1 − kδ)ε_0
    Run DistSeparation(f, ε_k, δ) c_2 log(k + 1) times
    Let M be a majority vote on the outputs
    If M = "ε_f > ε_k"
        Return ε_k
```

Figure 5: DistApprox$_2$

on δ (which we consider to be a constant in Theorem 3.1) is a multiplicative factor of $poly(1/\delta)$.

Note that amplifying the probability of success cannot be achieved by simply repeating the algorithm enough times and taking a majority vote on the returned interval: a correct interval is not even uniquely determined. The final probability of success is actually boosted by choosing larger values for c_1, c_2. Replacing the constants c_1, c_2 with $\alpha c_1, \alpha c_2$ for some $\alpha > 0$ decreases the probability of failure by a factor of $e^{-\Omega(\alpha)}$ and increases the running time by a factor of at most α.

3.4 A Faster Algorithm for Small Distances

We show in this section how to slightly improve the query complexity of the algorithm to

$$O(\min\{\log\log \varepsilon_f^{-1}, \log\log\log n\}\, \varepsilon_f^{-1} \log n).$$

The running time is now expected (over the random bits used by the algorithm). To do this, we need the following theorem:

Theorem 3.4 *There exists an algorithm that computes a value ε such that $\varepsilon_f \in [\Omega(\varepsilon/\log n), \varepsilon]$ with probability at least $2/3$. The expected running time of the algorithm is $O(\varepsilon_f^{-1} \log n)$.*

Using this theorem, it is clear that the factor $\log\log\varepsilon_f^{-1}$ in the distance estimation algorithm can be replaced by $\min\{\log\log\varepsilon_f^{-1}, \log\log\log n\}$. Indeed, instead of taking $k = 1, 2, 3, \ldots$, and running the separation oracle for each value of ε_k a number of times (ie, $c\log(k+1)$ times), we redefine ε_k to be $(1/2-\delta)^k\varepsilon$, where ε is the estimate returned by Theorem 3.4. Because the maximum value of k is now $O(\log\log n)$, the expected running time drops to $O(\min\{\log\log\varepsilon_f^{-1}, \log\log\log n\}\varepsilon_f^{-1}\log n)$.

To prove Theorem 3.4, we turn to a construction introduced by Goldreich et al. [8]. Define a subset P of pairs of integers: $(i,j) \in P$ if $j > i$, and $j - i$ is at most t, where t is the largest power of 2 that divides either i or j. This set has the following two properties:

1. $|P| = \Theta(n\log n)$.
2. For any $i < j$, there exists k $(i < k < j)$ such that both $(i,k) \in P$ and $(k,j) \in P$. This means, in particular, that for any violation (i,j) of f, there exists a "witness" (i,k) or (k,j) of the violation in the subset P.

Now, for a function f, let M be a maximum matching in the violation graph (the undirected graph whose vertex set is $\{1,\ldots,n\}$ and where i is connected to j if $i < j$ and $f(i) > f(j)$). It is known [8] that $|M| = \Theta(\varepsilon_f n)$; to be precise, $\frac{1}{2}\varepsilon_f n \le |M| \le \varepsilon_f n$. Let $Q \subseteq P$ be the set of violations of f in P. Consider the bipartite graph $G = (V_1, V_2, E)$ defined as follows : Let $V_1 = M$ and $V_2 = Q$, and let the set of edges E join $(i,j) \in V_1$ with $(a,b) \in V_2$ if $\{i,j\} \cap \{a,b\} \neq \emptyset$.

By the second property above, and from the definition of a maximum matching, every vertex in V_2 has degree either 1 or 2, and every vertex in V_1 has degree at least 1; therefore, $|V_2| = \Omega(|M|)$. We would like to show that it is $O(|M|\log n)$. If we could do that, then by sampling from P and checking for violations, we could then estimate the size of Q and get the desired approximation. Unfortunately, it is not quite the case that the cardinality of the right side is always $O(|M|\log n)$. To fix this problem, we need to introduce some more randomness.

We slightly change the definition of P: for an integer $r \in [1,n]$ let P_r denote the subset of pairs defined as follows: $(i,j) \in P_r$ if $j - i$ is at most t, where t is the largest power of 2 that divides either $i + r$ or $j + r$. The set P_r still has the two properties above. In addition, if r is chosen uniformly at random then, for any i, the expected number of j such that $(i,j) \in P_r$ and j' such that $(j',i) \in P_r$ is $O(\log n)$.

Consider the graph G_r, defined exactly as G using P_r instead of P. The expected number of edges of the corresponding bipartite graph G_r, therefore, is $O(|M| \log n)$. So the expected cardinality of the right side is $\alpha|P_r|$, where $\alpha \in [\Omega(\varepsilon_f / \log n), O(\varepsilon_f)]$. We sample P_r to form an estimation $\hat{\alpha}$ for α and return $\varepsilon = C\hat{\alpha} \log n$, for some large enough constant C, to prove Theorem 3.4. The estimation follows the predictable scheme:

1. Pick a random $r \in \{1, \ldots, n\}$.
2. Pick a pair (i, j) uniformly[1] at random from P_r.
3. if (i, j) is a violation of f, output `success`, otherwise `failure`.

The success probability is precisely α, so repeating the sampling enough times sharpens our estimation to the desired accuracy, as indicated by the following fact.

Lemma 3.5 *Given a 0/1 random variable with expectation $\alpha > 0$, with probability at least $2/3$, the value of $1/\alpha$ can be approximated with a relative constant error by sampling it $O(1/\alpha)$ times on average. Therefore, α can be approximated within the same error and the same expected running time.*

Proof: Run Bernoulli trials on the random variable and define Y to be the number of trials until (and including) the first 1. It is a geometric random variable with $\mathbf{E}[Y] = 1/\alpha$, and $\mathbf{var}(Y) = (1-\alpha)/\alpha^2 \leq (\mathbf{E}[Y])^2$. By taking several samples of Y and averaging we get an estimate $\hat{1/\alpha}$ of $1/\alpha$. Using Chebyshev's inequality, a constant number of samples suffices to get a constant factor approximation.

□

4 Lower Bounds

We now show that the running time of our tolerant property tester (DistApprox) is almost optimal.

Lemma 4.1 *Any property tester for monotonicity requires $\Omega(\varepsilon_f^{-1} \log(\varepsilon_f n))$ queries to f.*

[1] Refer to [8] for a method to approximately pick a pair uniformly, which suffices for our needs.

Proof: Choose the values ε_f, n such that $\varepsilon_f n = 2^t$ for some $t \in \mathbb{N}$ (where c is some small absolute constant). We will show that any randomized algorithm, that given a function $f : [n] \to \mathbb{R}$ ε_f-far from being monotone, outputs a violation to monotonicity (outputs two integers $i < j$ such that $f(i) > f(j)$) with probability $> 2/3$, requires expected $\Omega(\varepsilon_f^{-1} \log \varepsilon_f n)$ time. We will construct a distribution $\mathcal{F}$ of ε_f-far functions, and show that any *deterministic* algorithm that finds with probability $> 1/3$ a violation on an input drawn from $\mathcal{F}$ runs in expected time of at least $\Omega(\varepsilon_f^{-1} \log \varepsilon_f n)$. By Yao's minimax lemma [12], the lemma is proved.

We construct $\mathcal{F}$. Given $1 \leq r \leq \log(\varepsilon_f n)$ and $0 \leq k \leq (2\varepsilon_f)^{-1} - 1$, we define[2] a function f_r^k as follows: Divide the interval $[2k\varepsilon_f n, (2k+2)\varepsilon_f n - 1]$ into 2^r subintervals of length $\ell_r = 2\varepsilon_f n / 2^r$, each of the form $\mathcal{I}_j = [2k\varepsilon_f n + j\ell_r, 2k\varepsilon_f n + (j+1)\ell_r - 1]$ $\forall\ 0 \leq j \leq 2^r - 1$. Let $\mathcal{I} = \bigcup_{j\ odd} \mathcal{I}_j$. Then, (refer to Figure 6)-

$$f_r^k(i) = \begin{cases} i - \ell_r & \text{if } i \in \mathcal{I} \\ i & \text{otherwise} \end{cases}$$

The input distribution $\mathcal{F}$ is formed by choosing one of the f_r^k's uniformly at random. It is not hard to see that all of these functions are ε_f-far from being monotone. Associate with each function $f \in \mathcal{F}$ its set of violating pairs S_f. Note that all the S_f's are disjoint. Let us focus on the comparison model. The problem of determining a violating pair is essentially that of finding an element from a universe of size $\Omega(\varepsilon_f^{-1} \log \varepsilon_f n)$, from which the element is chosen uniformly at random. Since the deterministic algorithm succeeds with probability $> 1/3$ in finding the needle in haystack, it requires expected $\Omega(\varepsilon_f^{-1} \log \varepsilon_f n)$ comparisons between paris of f-values. Note however that any t queries to f could give rise to as many as $\binom{t}{2}$ comparisons, weakening our lower bound. The structure of the functions in $\mathcal{F}$ does not allow this: indeed, any t queries to f give rise to comparisons between pairs belonging to at most $O(t)$ sets S_f for $f \in \mathcal{F}$. Therefore, the lower bound holds also with respect to the number of *queries* to f. This concludes the proof.

□

5 The Higher Dimensional Case

We consider the generalization of monotonicity testing and distance estimation to functions on the d-dimensional hypercube. More precisely, our input

[2]For clarity, we shall assume that the domain is $\{0, 1, \cdots, n-1\}$.

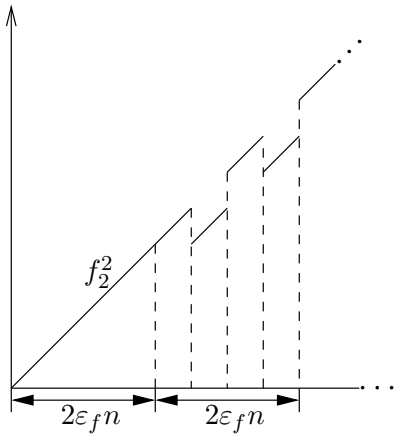

Figure 6: *Lower Bound for Testing: The function* f_2^2

is now a function $f : [n]^d \to \mathbb{R}$. The function f is monotone if $f(\mathbf{x}) \leq f(\mathbf{y})$ for every $\mathbf{x}, \mathbf{y} \in [n]^d$ such that $\mathbf{x} \leq \mathbf{y}$, where the partial order $\leq$ denotes the d-fold product of the linear order on $[n]$ ($\mathbf{x} \leq \mathbf{y}$ if $\mathbf{x}_i \leq \mathbf{y}_i$ for $i = 1, \ldots, d$). The distance ε_f of f from monotonicity is the minimal fraction of values that need to be changed in order to make it monotone with respect to $\leq$. This generalization has been well studied [9, 10, 13]. We will use a simple technique to transfer the one dimensional result to the higher dimension case, and obtain Theorem 1.1.

We start with some definitions. For $j = 1, \ldots, d$, we let $\leq_j$ denote the projected partial order on $[n]^d$: $\mathbf{x} \leq_j \mathbf{y}$ if $\mathbf{x}_j \leq \mathbf{y}_j$ and $\mathbf{x}_{j'} = \mathbf{y}_{j'}$ for $j' \in [d] \setminus \{j\}$. It is not hard to see that f is monotone w.r.t. $\leq$ if and only if it is monotone with respect to $\leq_j$ for $j = 1, \ldots, d$. Let $\varepsilon_{f;j}$ denote the distance of f from monotonicity w.r.t. $\leq_j$. Interestingly, up to a factor of $O(2^d)$, ε_f can be bounded by a linear combination of the $\varepsilon_{f;j}$'s. More precisely, we have the following inequalities:

Lemma 5.1

$$d\varepsilon_f \geq \sum_{j=1}^{d} \varepsilon_{f;j} \geq \varepsilon_f / 2^{d+1} \ .$$

The left side of the inequality is trivial, because $\varepsilon_f \geq \varepsilon_{f;j}$ for all j. The right side was proven by Ailon et al. in [1], slightly improving an independent result by Halevy et al. in [11].

We now generalize the notion of δ-big: Let $\mathbf{x} \in [n]^d$ and $j \in [d]$. We say that the point-direction pair $(\mathbf{x}, j)$ is δ-big if there exists $\mathbf{y} >_j \mathbf{x}$ such that

$$\left|\left\{ \mathbf{x} \leq_j \mathbf{z} \leq_j \mathbf{y} \,|\, f(\mathbf{z}) < f(\mathbf{x}) \right\}\right| \geq (1/2 - \delta)(\mathbf{y}_j - \mathbf{x}_j + 1)$$

or, similarly, $\mathbf{y} <_j \mathbf{x}$ such that

$$\left|\left\{ y \leq_j \mathbf{z} \leq_j \mathbf{x} \,|\, f(\mathbf{z}) > f(\mathbf{x}) \right\}\right| \geq (1/2 - \delta)(\mathbf{x}_j - \mathbf{y}_j + 1).$$

By Lemma 2.3 summed over all directions, we have

Lemma 5.2 *(i) At least $\sum_{j=1}^{d} \varepsilon_{f;j} n^d$ point-direction pairs are 0-big; (ii) No more than $(2 + 4\delta/(1-2\delta)) \sum_{j=1}^{d} \varepsilon_{f;j} n^d$ point-direction pairs are δ-big.*

Combining this with Lemma 5.1, we have:

Lemma 5.3 *(i) At least $\varepsilon_f n^d / 2^{d+1}$ point-direction pairs are 0-big; (ii) No more than $(2 + 4\delta/(1-2\delta)) d \varepsilon_f n^d$ point-direction pairs are δ-big.*

Proof: **[Theorem 1.1]** For $k = 0, 1, \ldots,$ run DistSeparation$c \log(k+2)$ times with f, $\varepsilon = ((1/2 - \delta) d^{-1} 2^{-d-1})^k$ and δ, until the algorithm reports $\varepsilon_f > \varepsilon_l$. We now know that ε_f is enclosed in an interval of the form $[\varepsilon_{l+1}, \varepsilon_{l-1}]$, which we shrink in the same manner as we did for the one-dimensional case. The total running time is $O(2^d d \varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$, as required. There is one additional detail that must be defined: how do we redefine DistSeparationfor a d-dimensional function? Instead of picking s random $i \in [n]$, we pick s random point-direction pairs $(\mathbf{y}, j) \in [n]^d \times [d]$. We define

$$x_{2k-1}^{(\mathbf{y},j)} = \alpha[\mathbf{y}, \mathbf{y} + (1+\delta/4)^k \mathbf{e}^{(j)}]$$
$$x_{2k}^{(\mathbf{y},j)} = \beta[\mathbf{y} - (1+\delta/4)^k \mathbf{e}^{(j)}, \mathbf{y}] \ ,$$

where $\mathbf{e}^{(j)} \in \{0,1\}^d$ is 0 everywhere except for a single 1 in coordinate j. The random variable $\alpha[\mathbf{y}, \mathbf{y}']$ for $\mathbf{y}' \geq_j \mathbf{y}$ is now defined by picking a random $\mathbf{z}$ between (inclusive) $\mathbf{y}$ and $\mathbf{y}'$ and returning 1 if $f(\mathbf{y}) > f(\mathbf{z})$, otherwise 0. The random variable β is defined similarly for the other direction. □

Acknowledgements

We would like to thank the anonymous referees for various helpful comments and suggestions.

References

[1] Ailon, N., Chazelle, B. *Information Theory in Property Testing and Monotonicity Testing in Higher Dimension* ECCC report TR04-068.

[2] Batu, T., Rubinfeld, R., White, P. *Fast approximate PCPs for multidimensional bin-packing problems*, Proc. RANDOM (1999), 245–256.

[3] Dodis, Y., Goldreich, O., Lehman, E., Raskhodnikova, S., Ron, D., Samorodnitsky, A. *Improved testing algorithms for monotonicity*, Proc. RANDOM (1999), 97–108.

[4] Ergun, F., Kannan, S., Kumar, S. Ravi, Rubinfeld, R., Viswanathan, M. *Spot-checkers*, Proc. STOC (1998), 259–268.

[5] Fischer, E. *The art of uninformed decisions: A primer to property testing*, Bulletin of EATCS, 75: 97-126, 2001.

[6] Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., Samorodnitsky, A. *Monotonicity testing over general poset domains*, Proc. STOC (2002), 474–483.

[7] Goldreich, O. *Combinatorial property testing - A survey*, in "Randomization Methods in Algorithm Design," 45-60, 1998.

[8] Goldreich, O., Goldwasser, S., Lehman, E., Ron, D., Samordinsky, A. *Testing monotonicity*, Combinatorica, 20 (2000), 301–337.

[9] Goldreich, O., Goldwasser, S., Ron, D. *Property testing and its connection to learning and approximation*, J. ACM 45 (1998), 653–750.

[10] Halevy, S., Kushilevitz, E. *Distribution-free property testing*, Proc. RANDOM (2003), 302–317.

[11] Halevy, S., Kushilevitz, E. *Testing Montonicity over Graph Products*, ICALP (2004).

[12] Motwani, R., Raghavan, P. *Randomized Algorithms*, Cambridge University Press (1995), Chapter 2, 34–35

[13] Parnas, M., Ron, D., Rubinfeld, R. *Tolerant property testing and distance approximation*, ECCC 2004.

[14] Ron, D. *Property testing*, in "Handbook on Randomization," Volume II, 597-649, 2001.

[15] Rubinfeld, R., Sudan, M. *Robust characterization of polynomials with applications to program testing*, SIAM J. Comput. 25 (1996), 647–668.