6.889 Sublinear Time Algorithms April 24th, 2019

Lecture 20

Lecturer: Ronitt A. Rubinfeld *Scribe: Theodore Katz*

Topics Covered

- Linear functions, k-linearity
- Communication complexity
- Lower bounds for property-testing linearity, by reduction from communication complexity

1 Linear Functions (Homomorphisms)

1.1 Defining Linear Functions

Definition 1 *A function f is said to be* linear *if for all x and y, $f(x+y) = f(x)+f(y)$.*

Today, we consider the linearity of functions from bitstrings to bits: $f : \{0,1\}^d \to \{0,1\}$. We define the "+" operator as bitwise XOR, i.e. a function is linear if for all x and y, $f(x \oplus y) = f(x) \oplus f(y)$.
Examples:

- $f(x) = 0$ is linear.
- $f(x) = 1$ is not linear. Note that for arbitrary x and y, $f(x \oplus y) = 1$ but $f(x) \oplus f(y) = 1 \oplus 1 = 0$.

- Given any bitvector b, $f(x) = \bigoplus_{i=1}^{d} x_i b_i$ (the dot product of x and b) is linear. In fact, this encompasses all linear functions over this domain, range, and "+" operator.

Fact 2 *Linear functions are parity functions*

A function f is linear iff $f(x) = \bigoplus_{i \in S} x_i$ for some set $S \subseteq [d]$. Equivalently, f is linear iff $f(x) = \bigoplus_{i=1}^{d} x_i b_i$ for some bitvector $b \in \{0,1\}^d$.

1.2 k-linearity

Definition 3 *k-linearity*

Given an integer k, a function f is said to be *k-linear* if f is linear, and f depends on exactly k specific bits of the input. Equivalently, f is k-linear iff $f(x) = \bigoplus_{i \in S} x_i$ for some set $S \subseteq [d]$ such that $|S| = k$.

2 Testing for Linearity

Given query access to a function $f : \{0,1\}^d \to \{0,1\}$ (with domain size $n = 2^d$), our goal is to distinguish the case where f is linear from the case where f is ϵ-far from linear.

Definition 4 *ϵ-far from linear*

A function f is said to be *ϵ-far* from another function g if their output differs on at least ϵ fraction of possible inputs. A function f is said to be *ϵ-far from linear* if for all linear functions g, f is ϵ-far from g. Similarly, f is *ϵ-far from k-linear* if for all k-linear functions g, f is ϵ-far from g. Intuitively, f is ϵ-far from (k-)linear if at least ϵn outputs of f would need to be changed in order to make it (k-)linear.

2.1 First Approach: Learning-Based Algorithm

Theorem 5 *A property testing algorithm can test for linearity (or k-linearity) in* $O\left(d + \frac{1}{\epsilon}\right)$ *queries.*

Consider the following algorithm:

1. Query f on the zero vector. If $f(0^d) \neq 0$, output "fail" and halt.
2. For all i in $[d]$, let $e_i = f(0^{i-1}10^i)$ (i.e. query f on the bitstring with only the ith bit set). Let S be the set of values i for which $e_i = 1$.
3. For $\frac{1}{\epsilon}$ random bitstrings x, check if $f(x) = \bigoplus_{i \in S} x_i$. If this equality holds for all sampled x, output "pass"; otherwise output "fail".

Trivially, this algorithm outputs "pass" for all linear functions. If f is ϵ-far from linear, with high probability it will find one of the bad inputs and correctly output "fail".

This algorithm only tests for linearity, but it can easily be converted to test for k-linearity by simply failing after step 2 if $|S| \neq k$.

Runtime: $O\left(d + \frac{1}{\epsilon}\right)$ queries. Can we remove the dependence on d?

2.2 Non-Learning-Based Algorithm

Theorem 6 *A property testing algorithm can test for linearity in* $O\left(\frac{1}{poly(\epsilon)}\right)$ *queries.*

Consider the algorithm that simply picks two random bitstrings x and y, and verifies that $f(x) \oplus f(y) = f(x \oplus y)$.

With high probability, the algorithm will output "fail" for functions that are ϵ-far from linear after $O\left(\frac{1}{\text{poly}(\epsilon)}\right)$ trials. (This will be proved in a future lecture.)

However, this algorithm cannot determine whether f is k-linear for a given k. In fact, the learning-based approach is known to be optimal for testing k-linearity in terms of the dependence on d.

Claim 7 *Distinguishing functions that are k-linear from functions that are ϵ-far from k-linear requires $\Omega(d)$ queries.*

We will prove a weaker version of this claim (a lower bound of $\Omega(k)$ queries) by reducing it from another known-hard problem in communication complexity.

3 Communication Complexity (Shared-Randomness Setting)

In a *communication complexity* setting, Alice knows an input $x = x_1...x_d$ and Bob knows an input $y = y_1...y_d$. Alice and Bob can send messages to each other. Their goal is to compute $f(x, y)$ for some known function f, while minimizing the total number of bits sent to each other.[1]

Today, we we are assuming Alice and Bob are in a *shared-randomness* setting, so they can both access the same pool of random bits without any communication cost. Since we are using randomized protocols, we are satisfied with computing $f(x, y)$ correctly with high probability.

Examples:

- $f(x,y) = \left(\bigoplus_{i=1}^{d} x_i\right) \oplus \left(\bigoplus_{i=1}^{d} y_i\right)$ (the XOR of all of the bits in x and y) can be computed with 1 bit of communication. Alice XORs all the bits of x together and sends the result to Bob, then Bob XORs it with all the bits of y.
- $f(x,y) = \left(\sum_{i=1}^{d} x_i\right) + \left(\sum_{i=1}^{d} y_i\right)$ (the total number of '1' bits in x and y) can be computed with $O(\log d)$ bits of communication. Alice counts all of the '1' bits in x and sends the result to Bob, then Bob adds it to the number of '1' bits in y.

[1] In communication complexity more broadly, it's sometimes interesting to try to minimize other things, such as the total number of messages sent, but today we're only concerned about minimizing the total number of bits sent.

- $f(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$ requires $\Theta(\log d)$ bits of communication.
- $f(x,y) = \begin{cases} 0 & \text{if there is some } i \text{ such that } x_i = y_i = 1 \\ 1 & \text{otherwise} \end{cases}$ (set disjointness) requires $\Theta(d)$ bits of communication.

4 Reducing Disjointness Testing To k-linearity testing

Consider the function $f(x,y) = \overline{\bigvee_{i=1}^{d} (x_i \wedge y_i)}$ which tests whether the sets of '1' bits in x and y are disjoint. Computing $f(x,y)$ with a communication complexity protocol requires $\Omega(k)$ bits of communication, where k is the number of '1' bits in x and y. We will use this fact today without proving it.

We will prove a lower bound for k-linearity testing as follows: Given a k-linearity tester that makes q queries, we will construct a communication protocol that solves the set disjointness problem with $O(q)$ bits of communication. Since solving the set disjointness problem is known to require $\Omega(d)$ bits of communication, it must be the case that any k-linearity tester makes $\Omega(d)$ queries.

4.1 Communication Setup

Consider the following communication protocol. As usual, Alice has an input x and Bob has an input y. Define the set $A = \{i \mid x_i = 1\}$ to contain the locations of '1' bits in x, and similarly define $B = \{i \mid y_i = 1\}$ to contain the locations of '1' bits in y. Assume $|A| = |B| = k$.

Define the functions $f(z) = \bigoplus_{i \in A} z_i$, $g(z) = \bigoplus_{i \in B} z_i$, and $h(z) = f(z) \oplus g(z)$.

4.2 Relating the Communication Setup to Disjointness

Consider an example where $d = 4$, $k = 2$.

Suppose A, B are disjoint; say $A = \{1, 2\}$ and $B = \{3, 4\}$. Then $f(z) = z_1 \oplus z_2$ and $g(z) = z_3 \oplus z_4$, so $h(z) = z_1 \oplus z_2 \oplus z_3 \oplus z_4$. Note that h is 4-linear, since it depends on all 4 of the input bits.

Now instead suppose A, B are not disjoint; say $A = \{1, 2\}$ and $B = \{1, 3\}$. Then $f(z) = z_1 \oplus z_2$ and $g(z) = z_1 \oplus z_3$, so $h(z) = z_1 \oplus z_2 \oplus z_1 \oplus z_3 = z_2 \oplus z_3$. Note that since A and B both contain 1, the z_1 terms cancel out. Now h is only 2-linear, since it depends only on z_2 and z_3.

We can generalize this example as follows:

Fact 8 *If A and B are disjoint, then h is $2k$-linear. Otherwise, h is at most $2k - 2$-linear.*

4.3 Communication Protocol

Given a tester for k-linearity, we can now create a protocol that tests set disjointness. The protocol works by having Alice and Bob both simulate the k-linearity tester in sync, to determine whether the function h is $2k$-linear. Whenever the k-linearity tester would select a random value z, Alice and Bob use their shared randomness pool to select the same value of z and remain in sync. Whenever the k-linearity tester would query h on a value z, Alice computes $f(z)$ and sends the result to Bob, and Bob computes $g(z)$ and sends the result to Alice. Then Alice and Bob both know $f(z)$ and $g(z)$, so they can both compute the correct value of $h(z)$ and continue running the algorithm.

If the k-linearity tester requires q queries to h, then this protocol requires $2q$ bits of communication between Alice and Bob (since each query to h causes Alice to send one bit to Bob, and Bob to send one bit to Alice). Since A and B are disjoint iff h is $2k$-linear, this protocol correctly determines whether A and B are disjoint.

But it is known that any communication protocol to test disjointness requires at least $\Theta(k)$ bits of communication. Therefore, $2q \geq \Theta(k)$, so any algorithm to test for k-linearity must make at least $\Theta(k)$ queries.