ECS 189A Sublinear Algorithms for Big Data **Fall 2024**

Lecture 4: List Monotonicity in Hamming and ℓ^1 Distance

Lecturer: Jasper Lee *Scribe: Yuan Luo and Matt Stone*

Recall that property testing involves

- a set of objects $\mathcal{C}$
- a subset of objects $\mathcal{P} \subseteq \mathcal{C}$. This subset is called a property, and we say that an object $\mathcal{O}$ *has the property* $\mathcal{P}$ if $\mathcal{O} \in \mathcal{P}$
- a notion of distance between objects $d : \mathcal{C} \times \mathcal{C} \to [0, 1]$ (d can be a metric, pseudometric, etc). For a scalar $\epsilon > 0$, we say that an object $\mathcal{O}$ is *ϵ-far* from having the property $\mathcal{P}$ if $d(\mathcal{O}, \mathcal{P}) \geq \epsilon$

Given $\mathcal{O} \in \mathcal{C}$, we want an algorithm to test whether $\mathcal{O}$ has the property $\mathcal{P}$ v.s. $\mathcal{O}$ is ϵ-far from having the property $\mathcal{P}$ with probability of success at least $\frac{2}{3}$. The case of $\mathcal{O}$ is not in either of the two cases is out of our consideration. The goal is to minimize queries and time complexity.

We now define the list monotonicity problem. Let $\mathcal{C}$ be the collection of length n lists of entries that are real numbers between 0 and 1, i.e. $\mathcal{C} = [0, 1]^n$. Property $\mathcal{P}$ is the set of non-decreasing lists. We will also use function notation $f : [n] \to [0, 1]$ to denote a list. When f outputs only 0 or 1, then the function f denotes a binary list.

We now give a distance function definition for lists. As we will see, the algorithm and complexity will change depending on the choice of distance.

Definition 4.1. The Hamming distance between two lists f_1 and f_2 is $d_H(f_1, f_2) = \frac{1}{n}|\{i \in [n] : f_1(i) \neq f_2(i)\}|$

With Hamming distance, f is ϵ-far from $\mathcal{P}$ means that we need to change at least ϵn elements of f to make it monotonic.

Definition 4.2. The ℓ^1 distance between two lists f_1 and f_2 is $\|f_1 - f_2\|_1 = \frac{1}{n}\sum_i |f_1(i) - f_2(i)|$

1 Property Testing for List Monotonicity in Hamming Distance

Some natural approaches are:

1. randomly sample linear fractions of data and test
2. divide + conquer to quicksort
3. query uniformly non-adaptively and check monotonicity
4. pick i randomly and query its neighbor $f(i) \leq f(i + 1)$ and repeat
5. pick a random pair or indices (i, j) and test whether $f(i) \leq f(j)$ and repeat.

These approaches are natural but can require linear query complexity. For example, Approach 4 has a lower bound of $\Omega(n)$ on its query complexity. This lower bound can be seen by noting that the list 11111…10…00000 requires $\Omega(n)$ queries until it sees the "10" transition, since the probability of seeing this transition is $\frac{1}{n}$. Note that this list is 1/2-far from monotonic under both notions of distance.

Similarly, Approach 5 requires $\Omega(n)$ queries. This lower bound can be seen by noting that the concatenation of small decreasing lists, let us say 321 654 987 …, requires that the two randomly chosen indices fall in the same decreasing sublist. This happens with probability $\frac{2}{n-1}$ and so this list requires $\Omega(\frac{1}{n})$ queries. Note that this list is at least 2/3-far from monotone under both notions of distance.

We will show an algorithm with an upper bound $\mathcal{O}(\frac{1}{\epsilon}\log n)$ and a lower bound of $\Omega(\log n)$ to test ϵ-far. For now, assume that the elements of an input list f are distinct.

Algorithm 1 Check list monotonicity in Hamming distance

```
1: Input: A list $f$ of length n.
2: Output: True if the algorithm accept $f$ is monotone and False otherwise.
3: Pick $O\left(\frac{1}{\epsilon}\right)$ indices $L \subseteq [n]$ uniformly.
4: for each index $i \in L$ do
5:    Run binary search for $f(i)$. Let the ending index be $j$.
6:    if $j \neq i$ then
7:       return False
8:    end if
9: end for
10: return True
```

We also introduce the following notation, which will be used in the proceeding proofs.

Definition. We say an index i of the list f is *good* if the binary search of $f(i)$ ends at i.

Figure 1 is an example of a list L without property $\mathcal{P}$ traversed by the binary search algorithm. The middle entry is queried and if $f(i)$ is less than this entry, then the algorithm recurses on the left half of the list. Otherwise, it recurses on the right half of the list. The good indices are highlighted in green and the others are highlighted in red.

Proposition 4.3. *For any monotonic list, Algorithm 1 will accept it.*

Proposition 4.4. *Algorithm 1 will reject all lists that are ϵ-far from sorted lists $\mathcal{P}$ with probability at least $\frac{2}{3}$.*

Proof. In Lemma 4.5, we will show that if a list f is ϵ-far, it has at most $(1-\epsilon)n$ many good elements. Note that if f is ϵ-far, Algorithm 1 fails when all query indices are good. Thus, the probability of its failure is bounded above by

$$\left(\frac{(1-\epsilon)n}{n}\right)^{O\left(\frac{1}{\epsilon}\right)} = (1-\epsilon)^{O\left(\frac{1}{\epsilon}\right)} \leq e^{-O(1)}$$

which is less than $\frac{1}{3}$ for all ϵ with appropriately chosen constant for the big-oh. □

The takeaway of the above argument is if there are not enough bad indices, then the list is close to monotonic anyway. These were the cases we were allowed to fail.

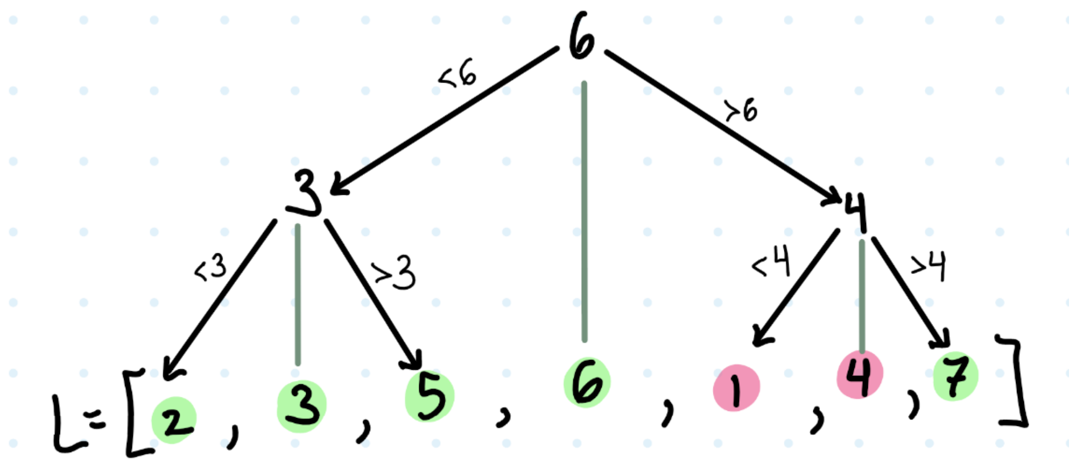


Figure 1: Binary search results of elements in a non-sorted list. Good indices are colored green and otherwise red. As we see in this example and prove in Lemma 4.6, the good indices form a monotonic sublist.

Lemma 4.5. *If an array f is ϵ-far, it has at most $(1-\epsilon)n$ many good elements.*

Proof. We will prove its contrapositive. Assume there are more than $(1-\epsilon)n$ good indices in f. We want to show the list f is $<\epsilon$-far from sorted.

As shown later in Lemma 4.6, those good indices form a monotone sublist of f. One way to fix f to make it monotone is to change f at the rest of the non-good indices. Concretely, because the set of good indices $I = \{i_1, i_2, ..., i_k\} \subset [n]$ gives a partition of $[n]$, we can set all elements between two indices i_j and i_{j+1} to $f[i_j]$.

By assumption, there are at most ϵn non-good indices. Thus, we have an upper-bound of distance from f to the monotone property $d(f, \mathcal{P}) = \inf_{\mathcal{O} \in \mathcal{P}}(f, \mathcal{O}) \leq \epsilon$. □

Lemma 4.6. *Good indices form a monotonic sublist.*

Proof. Consider a pair of good indices i, j with $i < j$. For each, consider their binary search path.

Let k be the last index shared by i and j's binary search path. Because i and j are good, there was no inconsistency detected, so $f(i) \leq f(k)$ and $f(k) \leq f(j)$ (see an example in Figure 2). Therefore $f(i) \leq f(j)$. □

Remarks:

- We can remove the distinctness assumption of the input list by creating a new list $f'(i) = (f(i), i)$ and use lexicographical ordering on this new list.
- Algorithm 1 is proximity-oblivious (ϵ-oblivious), i.e. it does not need to know ϵ to run
- An algorithm is *adaptive* if some query locations depend on other query results, i.e. an algorithm is *non-adaptive* algorithm if it can commit to all the query locations before the algorithm runs. Note that adaptive algorithms cannot be parallelized. Algorithm 1 is adaptive because of the binary search. It can, however, be made non-adaptive by querying not only $O(\frac{1}{\epsilon})$ indices $L \subseteq [n]$ but also, for each index $i \in L$,

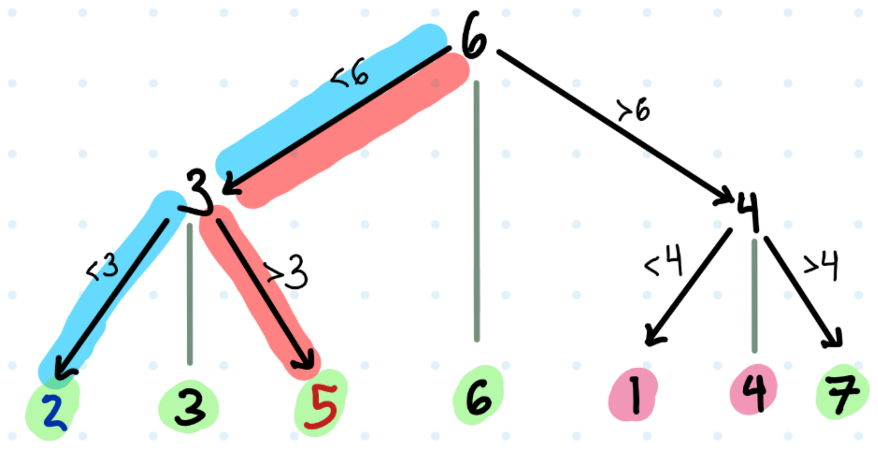


Figure 2: Binary search paths of good indices $i = 0$ and $j = 2$. Querying for indices $i = 0$ with $f(i) = 2$ and $j = 2$ with $f(j) = 5$ yields the blue and red search paths, respectively. The last index shared by these two paths is $k = 1$ with $f(k) = 3$.

querying all the entries along the binary search path that lead to the i^{th} entry in the list.

As we have seen, there is a non-adaptive algorithm for general $[0, 1]$ lists with query complexity $O(\frac{1}{\epsilon} \log n)$ that succeeds with probability at least $\frac{2}{3}$. In fact, the query complexity can be improved by further restricting the possible input lists:

Fact 4.7. *If we restrict to binary lists, there is a non-adaptive algorithm with query and time complexity $\mathcal{O}(poly(\frac{1}{\epsilon}))$ with probability $\frac{2}{3}$. Moreover, this algorithm is one-sided, suceeding on all monotonic lists. This will be proved on HW2.*

2 Comparing Hamming and ℓ^1 Distances

We now compare the two notions of distance between lists.

Proposition. *For any lists $f_1, f_2 : [n] \to [0, 1]$, we have $\|f_1 - f_2\|_1 \leq d_H(f_1, f_2)$. Equality is obtained when f_1 and f_2 are binary lists.*

Proof. Let f_1 and f_2 be two fixed lists and let I be the set of indices i on which f_1 and f_2 differ, that is,

$$I = \{i \in [n] : f_1(i) \neq f_2(i)\}$$

Then

$$\|f_1 - f_2\|_1 = \frac{1}{n} \sum_{i \in I} |f_1(i) - f_2(i)| \leq \frac{1}{n} \sum_{i \in I} 1 = \frac{1}{n} |I| = d_H(f_1, f_2)$$

Note that equality is obtained when $|f_1(i) - f_2(i)| = 1$ for all $i \in I$. □

We also want to compare how far a given list f is from having the the monotone list property $\mathcal{P}$ in Hamming and ℓ^1 distance.

Definition 4.8. Let $f : [n] \to [0,1]$ be a list and $\mathcal{P} \subset \mathcal{C}$ be the monotone list property. The Hamming distance between the function f and the property $\mathcal{P}$, denoted $d_H(f, \mathcal{P})$, is

$$\min_{g \in \mathcal{P}} d_H(f, g)$$

Definition 4.9. Let $f : [n] \to [0,1]$ be a list and $\mathcal{P} \subset \mathcal{C}$ be the monotone list property. The ℓ^1 distance between the function f and the property $\mathcal{P}$, denoted $\|f - \mathcal{P}\|_1$, is

$$\min_{g \in \mathcal{P}} \|f - g\|_1$$

In both cases, the min terminology is justified since the minimization is of a continuous function over a compact set and so the infimum is attained.

Finally, denote the set of monotone binary lists by $\mathcal{P}_{\text{bool}}$ ("bool" stands for boolean). As before, the set of all non-decreasing lists is denoted $\mathcal{P}$.

With these definitions, we have the following as corollary of the previous proposition.

Corollary 4.10. *For any lists $f : [n] \to [0,1]$, we have $d_H(f, \mathcal{P}) \geq \|f - \mathcal{P}\|_1$.*

Proof. If f has property $\mathcal{P}$, then both distances are zero and so the claim is shown. If f does not have property $\mathcal{P}$, let g be a function closest to f in $\mathcal{P}$ in the Hamming distance. Then

$$d_H(f, \mathcal{P}) = d_H(f, g) \geq \|f - g\|_1 \geq \min_{g \in \mathcal{P}} \|f - g\|_1$$

as desired. □

Proposition. *If $f \in \mathcal{P}_{bool}$, then $\|f - \mathcal{P}_{bool}\|_1 = d_H(f, \mathcal{P}_{bool}) \geq d_H(f, \mathcal{P}) \geq \|f - \mathcal{P}\|_1$*

This follow immediately from $\mathcal{P}_{\text{bool}} \subset \mathcal{P}$ and Corollary 4.10.

3 Property Testing for List Monotonicity in ℓ^1 Distance

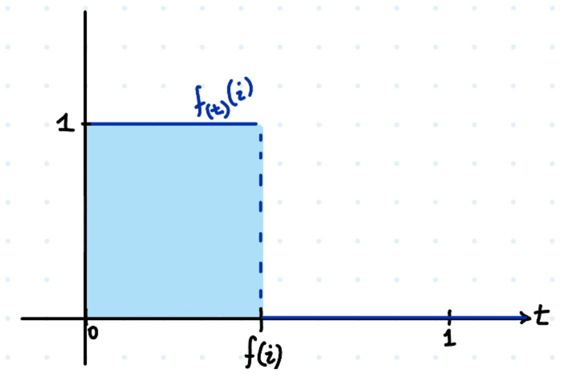
We will give a completely different algorithm to perform testing in ℓ^1 distance by reducing to binary list testing in Hamming distance, i.e. reduce $[0,1]^n$ testing to $\{0,1\}^n$ testing.

Definition 4.11. The indicator list f with threshold t, denoted $f_{(t)}$, is the list defined by

$$f_{(t)}(i) = \mathbb{1}_{\{f(i) > t\}}$$

Note that the indicator list with threshold t is a binary list, even if f is not itself a binary list.

Remark 4.12. For all $[0,1]$ lists, we have that $f(i) = \int_0^1 f_{(t)}(i)dt$. This is shown in the figure below:



The integral has exactly value $f(i)$ since this is the area under the curve $f_{(t)}(i)$.

It is also easy to see that

Proposition 4.13. *f is monotone if and only if $f_{(t)}$ is monotone for all $t \in [0, 1]$.*

Algorithm 2 Check list monotonicity in ℓ^1 distance

```
1: Input: A list $f$ of length n.
2: Output: True if the algorithm accept $f$ is monotone and False otherwise.
3: Query poly($\frac{1}{\epsilon}$) indices (non-adaptively according to Tester in Fact 4.7)
4: if the sublist formed by the queried indices is monotone then
5:     return True
6: else
7:     return False
8: end if
```

It is also true that

Lemma 4.14. $\|f - \mathcal{P}\|_1 = \int_0^1 \|f_{(t)} - \mathcal{P}\|_1 dt$

Proof. We first show that $\|f - \mathcal{P}\|_1 \leq \int_0^1 \|f_{(t)} - \mathcal{P}\|_1 dt$.

Let $g_{(t)}$ be a monotone function closest to $f_{(t)}$ in ℓ^1 distance. Then

$$\begin{aligned}
\|f - \mathcal{P}\|_1 &\leq \|f - g\|_1 && \text{Definition 4.9}\\
&= \frac{1}{n}\sum_{i\in[n]} |f(i) - g(i)| && \text{Definition 4.2}\\
&= \frac{1}{n}\sum_{i\in[n]} \left|\int_0^1 f_{(t)}(i) - g_{(t)}(i) dt\right| && \text{Remark 4.12}\\
&\leq \frac{1}{n}\sum_{i\in[n]} \int_0^1 \left|f_{(t)}(i) - g_{(t)}(i)\right| dt && \Delta \text{ Inequality}\\
&= \int_0^1 \frac{1}{n}\sum_{i\in[n]} \left|f_{(t)}(i) - g_{(t)}(i)\right| dt && \text{Linearity of } \int\\
&= \int_0^1 \|f_{(t)} - g_{(t)}\|_1 dt && \text{Definition 4.2}\\
&= \int_0^1 \|f_{(t)} - \mathcal{P}\|_1 dt && \text{Definition 4.9}
\end{aligned}$$

We now show that $\|f - \mathcal{P}\|_1 \geq \int_0^1 \|f_{(t)} - \mathcal{P}\|_1 dt$.

Let g be a monotonic function closest to f in ℓ^1 distance. Then

$$\begin{aligned}
\int_0^1 \|f_{(t)} - \mathcal{P}\|_1 dt &\leq \int_0^1 \left\|f_{(t)} - g_{(t)}\right\|_1 dt && \text{Definition 4.9} \\
&= \int_0^1 \frac{1}{n} \sum_{i \in [n]} \left|f_{(t)}(i) - g_{(t)}(i)\right| dt && \text{Definition 4.2} \\
&= \frac{1}{n} \sum_{i \in [n]} \int_0^1 \left|f_{(t)}(i) - g_{(t)}(i)\right| dt && \text{Linearity of } \int \\
&= \frac{1}{n} \sum_{i \in [n]} |f(i) - g(i)| && \text{Definition 4.2} \\
&= \|f - g\|_1 = \|f - \mathcal{P}\|_1 && \text{Definition 4.9}
\end{aligned}$$

□

We can now prove that Algorithm 2 works.

Theorem 4.15. *If f is ϵ-far from monotone, then Algorithm 2 will reject with probability at least $\frac{2}{3}$.*

Proof. If $\|f - \mathcal{P}\|_1 \geq \epsilon$, then there exists t s.t. $\|f_{(t)} - \mathcal{P}\|_1 \geq \epsilon$ by Lemma 4.14. By Corollary 4.10, $d_H(f_{(t)}, \mathcal{P}) \geq \|f_{(t)} - \mathcal{P}\|_1 \geq \epsilon$. Since $f_{(t)}$ is a binary list, we can consider running Tester 4.7 on $f_{(t)}$. Tester 4.7 is a one-sided tester, so with probability at least $\frac{2}{3}$, Tester 4.7 queries indices $i < j$ such that $1 = f_{(t)}(i) > f_{(t)}(j) = 0$. By Definition 4.11, $1 = f_{(t)}(i) > f_{(t)}(j) = 0$ if and only if $f(i) \geq t > f(j)$. Therefore, Algorithm 2 rejects f with probability at least $\frac{2}{3}$. □