CS 590 Sublinear Algorithms Jan 19, 2021

Lecture 1

Lecturer: Elena Grigorescu *Scribe: Tamalika Mukherjee*

1 Sublinear Algorithms: What and Why

The motivation behind developing sublinear algorithms comes from big data. Big data manifests from everything we do, some examples include the internet of things, sales transactions, web pages, health data, etc. This brings about a need for algorithm design that uses a small amount of time or space. Sublinear algorithms are usually randomized algorithms that look at a small portion of the data and output a result. There are three main types of sublinear models.

1. **Sublinear-time.** If data can be stored somewhere but there is little time available to read the data.
2. **Sublinear-space.** If data is too big to fit in memory, for example, streaming algorithms fit into this model.
3. **Sublinear-communication.** If data can be stored on multiple machines which communicate with each other.

2 Sublinear-time algorithms

1. **Approximation algorithms.** Algorithms that provide approximate solutions to computing some objective functions are called approximation algorithms. When this is done using only a small part of the input, this falls into the sublinear-time category. Some examples include computing the diameter of a graph, the number of connected components in a graph, and average degree of a graph.
2. **Property Testing.** This is also known as an approximate decision model. Informally, given an input, we want to check if this input has a specific property or is "far" from having it. Some properties for which there are efficient property testers are connectedness of graphs, 3-colorability, The property testing model was formalized by [4] and originated from program checking concepts in programming languages [3, 2].
3. **Locally Decodable/Testable Codes.** The object that we care about in this context is error-correcting codes. Typically, we encode data we want to save from errors like CDs, DVDs etc., using an error-correcting code like Reed Solomon. In the classical setting, we would receive the entire corrupted version of the code-word and we would apply some error-correcting algorithm to the entire word to recover the original message without errors. In the local model, we are only interested in particular bits of the codeword, since parsing the entire codeword would be $poly(N)$ time where N is the length of the codeword. A sublinear time decoding algorithm works as follows, describe a particular bit in $polylog(N)$ time, query a constant number of positions of the code word to correctly guess (with high probability) the specific bit at the i-th position (say). Some interesting questions that have been studied in this area are: (1) Given a code is it locally testable?

(2) Can each entry be corrected with high probability? (3) What is the best tradeoff between rate, distance and locality?

4. **Local Computation Algorithms.** This model is identical to that of Locally Testable codes, except the object of interest is more general, for example graphs, was introduced by [5]. A more concrete graph example in this model is finding whether a node belongs to the maximal independent set.

3 Sublinear Space Algorithms (Streaming algorithms)

In this model (introduced by [1]) we have a stream of data, and we want to compute the natural problems on this object, but with the constraint of limited memory. For example, if the data stream is graph related, then we might want to approximate the maximum matching, or vertex cover. Instead if the data stream is a bunch of numbers, we might be interested in the median or average.

4 Sublinear in Communication

In this model, given function $f : X \times Y \to Z$, where X, Y, Z are arbitrary sets, and two players Alice and Bob. Alice only knows $x \in X$ and Bob only knows $y \in Y$, the goal is to find the minimum possible bits required in communication to determine $f(x, y)$. For example $f : \mathbf{F}_2 \times \mathbf{F}_2 \to \mathbf{F}_2$ can be the XOR function, and $f(x, y) := x \oplus y$, in this case we need two bits of communication for Alice and Bob to determine $f(x, y)$. Many natural questions that we care about are hard in this model.

For example, consider the $\mathsf{EQUALITY}$ function, defined as $\mathsf{EQ} : \{0,1\}^N \times \{0,1\}^N \to \{0,1\}$,

$$\mathsf{EQ}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

It turns out that any deterministic communication protocol has to send *at least* N bits of communication to determine EQ, i.e., either one of the players needs to send their entire input to the other. Clearly, we cannot hope to achieve any sublinear algorithms in the context of EQ.

Another classical function is INDEX, denoted by $\mathsf{IDX} : \{0,1\}^N \times [N] \to \{0,1\}$, where

$$\mathsf{IDX}(\mathbf{x}, y) = x_y$$

This is another hard problem, where the complexity of both deterministic and randomized one-way communication is $\Omega(N)$.

The last example, we discuss here is set-disjointness, $\mathsf{DISJ} : \{0,1\}^N \times \{0,1\}^N \to \{0,1\}$, where $\mathbf{x}, \mathbf{y}$ are characteristic vectors of the sets in the def below,

$$\mathsf{DISJ}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{x} \cap \mathbf{y} = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Even the randomized communication complexity for set-disjointness is $\Omega(N)$, so again, we cannot hope to achieve sublinear algorithms for this case.

These communication complexity lower bounds are interesting because we can obtain lower bounds for other models from them. For example, lower bounds for communication imply lower bounds for streaming and property testing.

5 Deterministic 2-Approximation of Diameter (of a set of points)

For a distance metric D, the diameter of a set S is defined as $diam(S) := \max_{x,y \in S} D(x,y)$.

We will prove the following theorem.

Theorem 1 (Indyk) *Given a set S and distance metric D, there exists a deterministic algorithm that outputs d such that $diam(S)/2 \leq d \leq diam(S)$ in $O(|D|)$ time.*

Proof Note that since D is all the paired distances between the points in S, we have $O(|D|) = O(|S|^2)$. So if we get something linear in $|S|$, then it's sublinear in $|D|$.

Observe that for every x, y and every t, by triangle inequality,

$$\begin{aligned} D(x,y) &\leq D(x,y) + D(y,t) \\ &\leq 2\max\{D(x,t), D(y,t)\} \\ &\leq 2\max_{r \in S}\{D(r,t)\} \ . \end{aligned}$$

So the algorithm is as follows

1. Take any $t \in S$.
2. Output $d = \max_{r \in S}\{D(r,t)\}$

Clearly this is a linear-time algorithm (linear in $|S|$), and $d \leq diam(S)$. Also note that $diam(S) \leq 2d$. Thus combining these observations, we get the desired result. ■

6 More on Property Testing

We will first discuss property testing in more details and then see some basic results. A property is a collection of objects that all have a particular characteristic. For example, consider the set of all graphs on n vertices that are 3-colorable, i.e., $P_n = \{G_n : G_n \text{ is 3-colorable}\}$. We want to test whether an input is in the property or very far from the property. Given an input, we are interested in testing for membership in the specified property.

What do we mean by ε-far? This is defined in terms of hamming distance.

Definition 2 *Given functions $f, g : D \to R$, the relative hamming distance between f and g is*

$$\delta(f,g) := \frac{1}{|D|}|\{x : f(x) \neq g(x)\}|$$

Note that in this course, the size of domain D, will be finite. The distance of a function f to property P is $dist(f,P) = \min_{g \in P} \delta(f,g)$.

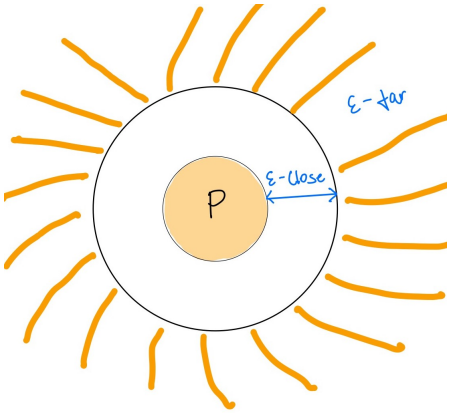

Figure 1: Illustration of Property Testing. Let the property of interest be denoted by the set P. If x is an input, then we are interested in the question of whether $x \in P$ or x is ε-far from P. Note that this is different from the decision question of whether $x \in P$ or $x \notin P$, since we do not care about the case when x lies in the ε-close region.

Definition 3 *Property P is k-locally testable if there exists a randomized algorithm A with black-box access to input such that*

1. *A makes k queries to the input.*
2. *(Completeness) If f is in P then A accepts.*[1]
3. *(Soundness) If f is ε-far from P then $\Pr[A \text{ accepts}] < 1/3$.*

6.1 Test if a function is the constant 1 function

For the property $P_n = \{f : f(x) = 1, \forall x \in [n], f : [n] \to \{0,1\}\}$, we interested in whether $f \in P_n$ or f is ε-far from P_n (i.e., f has at least an ε-fraction of 0's).

Claim 4 *P_n is $2/\varepsilon$-locally testable.*

The statement above is saying that for a fixed ε, we can distinguish between $f \in P_n$ and f being ε-far from P_n using only $2/\varepsilon$ many queries.

The tester A is as follows.

1. Pick $2/\varepsilon$ random x's (without repetition).
2. Compute $f(x_1, f(x_2), \ldots, f(x_{2/\varepsilon})$.
3. If there exists i such that $f(x_i) = 0$ then reject, otherwise accept.

The number of queries is exactly $2/\varepsilon$ which is constant with respect to n. We have to prove completeness and soundness.

[1] This condition for completeness is for a one-sided tester, we can also change this definition to a two-sided tester by accepting with probability $\geq 2/3$ instead of probability 1.

For completeness, note that if $f \in P_n$, then the tester never rejects. So the $\Pr[A \text{ accepts}] = 1$. For soundness, suppose that f is ε-far from P_n, i.e., f has $> \varepsilon n$ many zeroes. Then the probability that A accepts is the probability that no zero is hit in $2/\varepsilon$ trials, which is $\leq (1-\varepsilon)^{2/\varepsilon} \leq (e^{-\varepsilon})^{2/\varepsilon} = e^{-2} \leq 1/3$. Therefore $\Pr[A\text{accepts}] < 1/3$, as desired.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

[2] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 73–83, New York, NY, USA, 1990. Association for Computing Machinery.

[3] Manuel Blum and Sampath Kannan. Designing programs that check their work. *J. ACM*, 42(1):269–291, January 1995.

[4] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials withapplications to program testing. *SIAM J. Comput.*, 25(2):252–271, February 1996.

[5] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. *CoRR*, abs/1104.1377, 2011.