0368.416701 Sublinear Time Algorithms October 19, 2015

Lecture 1

Lecturer: Ronitt Rubinfeld *Scribe: Ido Begun, Yonatan Nakar, Yael Golan*

1 Introduction

This course will focus on the design of algorithms that are restricted to run in sublinear time, and thus can view only a very small portion of the data.

Traditionally, showing the existence of a linear time algorithm for a problem was considered to be the gold standard of achievement. However, if the input is too big, a linear time algorithm might be inadequate. The abundance of huge data sets that should be processed fast raises the need to develop algorithms whose running time is sublinear in the input size.

1.1 The small world property

We consider the following example: Suppose that we have a graph, whose nodes represent people and there is an edge between two people who know each other. The distance between two people is defined to be the shortest path between the two nodes in the graph. The diameter is the maximum distance between any pair. Given such a graph, we ask whether this graph has the "small world property", i.e., does it have a small diameter (namely, six degrees of separation as Milgram's experiment has suggested)? This is easily detectable after asking whether person A knows person B for all possible pairs of people. However, by the time all of these questions are fully answered, the answer might be very different, and moreover it might be too late to make use of the information. Fortunately, it might be feasible to construct algorithms based on random sampling which do not need to ask about all of the pairs.

1.2 What can we hope to do without viewing most of the data?

We cannot answer "for all" or "there exists" and other "exactly" type statements such as:

1. Are all individuals connected by at most six degrees of separation?
2. Exactly how many individuals on earth are left-handed?

But we might be able to answer the following kinds of questions:

1. Is there a large group of individuals connected by at most six degrees of separation?
2. Is the average pairwise distances of a graph roughly six?
3. Approximately how many individuals on earth are left-handed?

2 Sublinear time models

There are two models for sublinear time algorithms:

1. **Random Access Queries:**
   We can access any word of the input in one step.
2. **Samples:**
   We can get a sample of a distribution in one step. Alternatively, we can think about it as getting a random word of the input in one step.

In this lecture we only deal with examples of the first model.

3 Classical approximation problems

The following algorithm for *approximating the diameter of a point set* is the only deterministic algorithm we will see for *random access queries.* The other algorithms of this model will be randomized. However, In the *samples* model, we will see algorithms that are deterministic in the sense that there will not be any coins tossed by the algorithm itself but even in these algorithms there will be randomness. The source of that randomness is going to come from the sampling process, and not directly generated by the algorithm.

3.1 Approximate the diameter of a point set

Given: m points, described by a distance matrix D of size $n = m^2$, such that:

- D_{ij} is the distance from point i to point j
- D satisfies the triangle inequality: $\forall i, j, k. D_{ij} + D_{jk} \geq D_{ik}$
- D satisfies symmetry: $\forall i, j. D_{ij} = D_{ji}$

Output: Let d be the diameter of the point set. The algorithm should return k, l such that $D_{kl} \geq d/2$ (2-multiplicative approximation).

Algorithm 1

```
1: Pick $k$ arbitrarily.
2: Pick $l$ to maximize $D_{kl}$.
3: Return $k$ and $l$.
```

Time complexity: $\mathcal{O}(m) = \mathcal{O}(\sqrt{n})$.
Correctness: Let i, j be the points such that $D_{ij} = d$ and let k, l be the output of the algorithm.
Proof From the triangle inequality: $D_{ij} \leq D_{ik} + D_{kj}$.
Due to the way l was chosen: $D_{ki} \leq D_{kl}$ and $D_{kj} \leq D_{kl}$.
From the symmetry: $D_{ki} = D_{ik}$.
Therefore: $D_{ij} \leq D_{kl} + D_{kl} = 2D_{kl} \Rightarrow D_{kl} \geq d/2$ ■

4 Property testing

The *property testing* algorithms are kind of decision problems. We would like to know how to distinguish between an input that has a specific property from an input that does not have this property. Since it might take a long time to do it, we will try to find an approximation algorithm that can distinguish between inputs that have a specific property from those that are *far* from having this property. Namely:

- If the input has the property, the algorithm output should be *Yes.*
- If the input does not have the property but is *close* to having it, the algorithm output can be either *Yes* or *No.*
- If the input does not have the property and is *far* from having it, the algorithm output should be *No.*

In such problems, the following must be specified:

- The input representation (for example: whether a graph is represented by an adjacency matrix or an adjacency list).

- The definition of what we mean by *close* and *far* (for example: how many words can be different in order for the input to be still considered as *close* to the right one).

4.1 Sortedness of a sequence

Given: a list $y_1, y_2, \dots, y_n$.
Optimal output: whether the list is sorted or not.
This output requires n steps (linear time) in order to return a correct answer for sure since we must look at each y_i.
Our output: *quickly* decide whether the list is *close* to being sorted.
Definitions:

- Quickly is measured compared to the list size n. Our Goal - $\mathcal{O}(\log n)$.
- A list of size n is $\varepsilon - close$ to being sorted if we can delete at most εn values to make it sorted. Otherwise it is considered as $\varepsilon - far$ list.

We would like to find an algorithm that:

- Pass sorted lists.
- Fail on lists that are $\varepsilon - far$ from being sorted.
- The probability of success is greater than $\frac{3}{4}$. It means that the algorithm can fail both on sorted lists (will not happen in our algorithm) and on $\varepsilon - far$ from being sorted list with probability that is less than $\frac{1}{4}$.
  Notice that the value $\frac{3}{4}$ can increased by repeating the algorithm several times and outputting "fail" if it fails once or more and "success" otherwise).
- The time complexity will be $\mathcal{O}(\frac{1}{\varepsilon} \log n)$

4.1.1 First attempt

Algorithm 2

```
1: Pick random $i$ and test that $y_i \leq y_{i+1}$.
```

Even if we repeat the algorithm several times and return "fail" if at least one of the times fails and "success" otherwise, the algorithm is not good enough.
Example:

$$1, 2, 3, 4, 5, \dots, \frac{n}{4}, 1, 2, \dots, \frac{n}{4}, 1, 2, \dots, \frac{n}{4}, 1, 2, \dots, \frac{n}{4}$$

The success probability of the algorithm on the above example is $\frac{3}{n}$ (less than $\frac{3}{4}$). As a result, we need to repeat the algorithm $\Omega(n)$ times on this input. It happens even though the list is $\frac{3n}{4}$-*far* from being a sorted list because there are a few "break points" in it.

4.1.2 Second attempt

Algorithm 3

```
1: Pick random $i < j$ and test that $y_i \leq y_j$.
```

Here there is another algorithm that even if we repeat it several times and return "fail" if at least one of the times fails and "success" otherwise, the algorithm is not good enough.

Example:

$$4, 3, 2, 1, 8, 7, 6, 5, 12, 11, 10, 9, \ldots, 4k, 4k-1, 4k-2, 4k-3, \ldots$$

The success probability of the algorithm on the above example is less than $\frac{3}{4}$. It happens even though the largest monotone sequence is $\frac{n}{4}$. The algorithm must pick i, j from the same "group" in order to see the problem. According to the *birthday paradox*, it will take $\Omega(\sqrt{n})$ samples which is much bigger than $\mathcal{O}(\log n)$.

4.1.3 The final algorithm

A minor simplification: we assume the list is distinct - $\forall i \neq j . x_i \neq x_j$.
Such an assumption can always be made since every sequence $x_1, x_2, \ldots, x_n$ can be treated as $(x_1, 1), (x_2, 2), \ldots, (x_n, n)$
This transformation breaks ties without changing the order and even if $\exists i \neq j . x_i = x_j$, in the new transformation, $(x_i, i) \neq (xj, j)$.

Algorithm 4

```
1: for $\mathcal{O}(\frac{1}{\varepsilon})$ times do
2:     Pick random $i$.
3:     Look at the value of $y_i$.
4:     Do binary search for $y_i$.
5:     if the binary search find any inconsistencies then
6:         Return Fail.
7:     end if
8:     if we end up at location $i$ then
9:         Return Fail.
10:    end if
11: end for
12: Return Success.
```

Time complexity: $\mathcal{O}(\frac{1}{\varepsilon}\log n)$ since a binary search complexity on a list of size n takes $\mathcal{O}(logn)$ and we do it at most $\frac{1}{\varepsilon}$ times.
Correctness:
Definition: Index i is *good* if the binary search for y_i successful.
The algorithm can be restated as picking $\mathcal{O}(\frac{1}{\varepsilon})$ indices and returning *Success* if they are all *good*.
It is clear that if the list is sorted, all $i's$ are *good* and therefore the algorithm will always return *Success*.
We need to show that if the list is $\varepsilon - far$ from being sorted, the algorithm is likely to fail. We will prove the contrapositive: if the algorithm is likely to pass (output *Success*), the list is $\varepsilon -$ *close* to being sorted.
Proof If the algorithm is likely to pass, at least $(1-\varepsilon)n$ $i's$ are *good*.
The main observation is: *good* elements form increasing sequence.

- Let $i < j$ indices of two *good* elements. We will prove that $y_i < y_j$.

- Let k be the least common ancestor of i and j in the binary search.

- The search for i went left from k and the search for j went right from k, therefore $y_i < y_k < y_j \Rightarrow y_i < y_j$ as needed.

From the main observation we conclude that if at least $(1-\varepsilon)n$ $i's$ are *good*, the list is $\varepsilon -$ *close* to being sorted (the other $< \varepsilon n$ elements can be deleted and we will get a sorted list).
$\Rightarrow$ if the algorithm is likely pass, the list is $\varepsilon -$ *close* to being sorted. ■

4.2 Are all words distinct?

Given: input $x_1, x_2, \ldots, x_n$
Optimal output: whether all x_i distinct or not.
This output requires linear time in order to return a correct answer for sure since we must look at each x_i.
Our output: distinguish between three cases:

- All elements are distinct.
- The number of distinct elements is less than $(1-\varepsilon)n$.
- If neither case holds, algorithm can output arbitrarily.

The output does not depend on the objects' order, therefore we can sample.

Algorithm 5

```
1: Pick several independent samples.
2: if there are duplicates then
3:     Return Fail.
4: else
5:     Return Success.
6: end if
```

It's clear that if all elements are distinct, the algorithm will always output *Success*.
How many samples does the algorithm need to get?
For example, Consider the following input in a random order:

$$1, 1, 2, 2, 3, 3, \ldots, \frac{n}{2}, \frac{n}{2}$$

According to the *birthday paradox*, the algorithm needs only $\sqrt{n}$ samples in order to find a duplicate in this example. But are $\sqrt{n}$ samples enough for *any* input? In addition, the following issues need to be addressed:

- Do we sample with or without replacement?
- How to analyze the failure probability of picking two identical elements outputs *Fail*?
- How are the duplicates distributed?

Note: We can save the samples in a hash table in order that each compare will take constant time.
The idea: think about the duplicates in pairs (throw objects whose number of occurrences is odd).

5 Recall from probability courses

5.1 Markov's inequality

Let X be a positive random variable, and c be a positive real number, then:

$$Pr\left[X > c \cdot E(X)\right] < \frac{1}{c}$$

Where $E(X)$ is the expectation of X.

5.2 Chebyshev's inequality

Let X be a positive random variable, and c be a positive real number, then:

$$Pr\left[|X - E(X)| > c \cdot B\right] < \frac{1}{c^2}$$

Where B is a bound for the standard deviation of X.

5.3 Chernoff/Hoeffding Bounds

Let $X_1, X_2, ...X_n$ be independent 0/1 random variables.
Define $S := X_1 + X_2 + ... + X_n$, then:

$$Pr\left[S > (1+\delta) \cdot E(S)\right] < e^{\frac{-\delta^2 \cdot E(S)}{2}}$$

$$Pr\left[S < (1-\delta) \cdot E(S)\right] < e^{\frac{-\delta^2 \cdot E(S)}{3}}$$