0368.416701 Sublinear Time Algorithms Oct 19, 2009

Lecture 1

Lecturer: Ronitt Rubinfeld *Scribe: Daniel Shahaf*

1 Sublinear-time algorithms: motivation

Twenty years ago, there was practically no investigation of sublinear-time algorithms. One reason for this was that it wasn't that common to have extremely large datasets. Today, however, datasets of many different types may be so large that a linear-time algorithm would take ridiculously long. At other times, time constraints require us to make a decision too swiftly than to allow for examining the input in its entirety.[1]

In first part of the course we will model the input as written down somewhere such that we have query access to (i.e., for any i we can query for the ith bit of the input). Towards the end of the course, we will consider a different model where our inputs consists of samples taken from an unknown distribution.

Running in sublinear-time precludes us from reading the entire input; therefore, we will typically use sampling. Though sometimes we will use straightforward sampling, many of our algorithms will use more intricate algorithmic techniques combined with sampling.

Being sublinear-time will, in most cases, force us to use randomness in our algorithms and limit us to only hope for an approximate answer (in many cases getting a non-approximate answer requires reading the input fully). The next example is the only deterministic algorithm we will see in this course.

2 Examples

2.1 A deterministic algorithm

Example 1: point-set diameter

Given An $m \times m$ distance matrix d.

We assume the matrix is symmetric and satisfies the triangle inequality.

Goal Compute the diameter $\hat{d} \stackrel{\text{def}}{=} max_{u,v} d(u,v)$.

Algorithm

- Pick an arbitrary point x.
- Find the point y farthest from x.
- Output $z \stackrel{\text{def}}{=} d(x,y)$.

Analysis

Claim 2 *The algorithm's time complexity is sublinear.*

Proof The algorithm's time complexity is $O(m)$, i.e., $O(\sqrt{\text{input size}})$. ■

[1] Another reason for interest is the human quality of laziness: a quick answer that requires neither reading much input nor much computations appeals to many.

Claim 3 *The algorithm is a (multiplicative) 2-approximation algorithm for the diameter problem; that is,* $\hat{d}/2 \leq z \leq \hat{d}$.

Proof The right inequality is trivial. We show the left one.

Fix two points a, b such that the diameter is $\hat{d} = d(a, b)$. Then

$$\hat{d} = d(a, b) \leq d(a, x) + d(x, b) \leq d(x, a) + d(x, b) \leq d(x, y) + d(x, y) = 2z.$$

■

2.2 A decision problem

As we mentioned before, most sublinear algorithms must output some sort of approximation. In this example, we discuss a type of approximation that makes sense for outputs of decision problems.

Example 4: sequence monotonicity, attempt 1

Given An ordered list $X_1, \ldots, X_n$ of elements (with partial order '$\leq$' on them).

Goal Is the list monotone? That is, is $X_1 \leq \cdots \leq X_n$?

As stated, the goal requires looking at every single sequence element. (If we skip over even one of them, that one may be the only one breaking the monotonicity.) Therefore we relax the problem:

Example 5: sequence monotonicity, attempt 2

Given An ordered list $X_1, \ldots, X_n$ of elements (with partial order '$\leq$' on them) and a real fraction $\epsilon \in [0, 1]$.

Goal Is the list *close to monotone*?

(We will say that a list is ϵ-*close to monotone* if it has a monotone subsequence of length $(1 - \epsilon)n$.)

Required behavior We require a 2-sided (BPP) error:

- If the list is monotone, the test should pass with probability 3/4.
- If the list is ϵ-far from monotone, the test should fail with probability 3/4.

Remark The choice of 3/4 is arbitrary; any constant bounded away from 1/2 works equally well. We can amplify the definition from our constant to a different constant $1 - \beta$ by repeating our algorithm $O(\log \frac{1}{\beta})$ times and taking the majority answer.

Remark The behavior of the test on inputs that are very close to monotone, but are not monotone, is undefined. (Those inputs are ϵ'-close with $0 \lneq \epsilon' \leq \epsilon$.) This makes sense because those inputs are 'almost' monotone — so we allow ourselves the latitude to treat them as if they were monotone — while, all in all, they are *not* monotone, and declaring them as such is correct.

Here are a few algorithmic ideas that one might try to base a tester upon:

Idea 6: Pick $i < j$ randomly and test $x_i < x_j$.

We will show that this idea's complexity is $\Omega(\sqrt{n})$.

Fix some constant c, and consider the following sequence:

$$\underbrace{c, c-1, \ldots, 1}, \underbrace{2c, 2c-1, \ldots, c+1}, \ldots, \ldots, \ldots, \underbrace{n, n-1, \ldots, n-c+1}.$$

The longest monotone subsequence has length n/c (we can't pick twice from the same 'group' since each group is monotonically decreasing) — relatively small, so we would like this sequence to fail the test. We can see, however, that the test passes whenever it picks i, j from different groups.

The following can be shown:

If the test is repeated by repeatedly picking new pairs i, j, each time discarding the old pair, and checking each such pair independently of the others, then $\Omega(n)$ pairs are needed. However, if the test is repeated by picking k indices and checking whether the subsequence induced by them is monotone, then $\Theta(\sqrt{n/c})$ samples are needed (using the Birthday Paradox). We will see that we can do much better.

Idea 7: Pick i randomly and test $x_i \leq x_{i+1}$.

Fix some constant c, and consider the following sequence (of n elements):

$$\underbrace{1, 2, \ldots, n/c}, \underbrace{1, 2, \ldots, n/c}, \ldots, \underbrace{1, 2, \ldots, n/c}.$$

Again, the longest monotone subsequence has length $c + \frac{n}{c} - 1$ — relatively small, so we would like this sequence to fail the test. However, the test passes unless the i it picks is a "border point" (i.e., unless $X_i = n/c$), which happens with probability c/n. Therefore we expect to require a linear number of samples before detecting an input that should be rejected.

Idea 8: Combine the previous two ideas.

This would verify that the sequence is locally monotone, and also monotone at large distances, but would not verify that it is monotone in middle-range gaps. And counter-examples can be found. However, there exists a correct, $O(\log n)$-samples algorithms that works by testing pairs at various distances $1, 2, 4, 8, \ldots, 2^k, \ldots, n/2$.

Before giving an algorithm, we make the following assumption.

Assumption 9 *X_i are pairwise distinct.*

Proof Mentally replace each X_i by the tuple (X_i, i) and use dictionary order: to compare '(X_i, i)' to '(X_j, j)', compare the first coordinate and use the second coordinate to break ties. ■

Remark This trick does not hamper the sublinearity of the algorithm because it does not require any pre-processing; the transformation can be done on the fly as each element is accessed and compared.

Notation

'$[n]$' denotes the set $\{1, 2, \ldots, n\}$ of positive integers.

'$\in_{\mathrm{R}}$' denotes assignment of a random member of the set on its RHS to the variable on its LHS. If the distribution is not specified, it is the uniform distribution.

For example, '$x \in_{\mathrm{R}} [3]$' assigns to x one of the three smallest positive integers, chosen uniformly.

Algorithm

- Repeat $O(1/\epsilon)$ times:
  - Pick $i \in_{\mathrm{R}} [n]$.
  - Query (obtain) the value X_i.
  - Do binary search for X_i.
  - If either
    - an inconsistency was found during the binary search;
    - X_i was not found;

    then return fail.
- Return pass.

Inconsistency By an 'inconsistency' we mean the following: during the binary search, we maintain an interval of allowed values for the next value we query. The interval starts as '$[-\infty, +\infty]$. Its upper and lower bounds are updated whenever we take a step to the left (towards smaller elements) or to the right (towards larger elements), respectively. Whenever we query a value we assert that it is in the interval and raise an inconsistency if it isn't.

Time complexity This algorithm's time complexity is $O(\frac{1}{\epsilon} \log n)$, since the augmented binary search and the choosing of a random index cost $O(\log n)$ steps each; and those are repeated $O(1/\epsilon)$ times.

Correctness We will now show that the algorithm satisfies the required behavior. We will define which indices are 'good' and relate the number of bad indices to the length of a monotone sequence of elements at good indices.

Definition 10 *An index i is* good *if augmented binary search for i is successful (does not detect an inconsistency).*

Observation 11 *If $\geq \epsilon n$ indices are bad, then* $\mathrm{Prob}\,[\mathsf{pass}] < 1/4$.

Proof Let c be the constant under the '$O(1/\epsilon)$ repetitions' clause. Then

$$\mathrm{Prob}\,[\mathsf{pass}] \leq (1-\epsilon)^{c/\epsilon} \leq (1/\epsilon)^{c} < \frac{1}{4}, \tag{1}$$

where the last (strict) inequality follows by setting c to a large enough (constant) value. ■

Theorem 12 *The above algorithm has 2-sided error less than one quarter: it accepts good inputs with probability* 1 *and rejects bad inputs with probability at least* 3/4.

Proof If the list is monotone, then it passes with certainty because the binary search works and the X_i are assumed distinct. It remains to prove that far-from-monotone lists are rejected with high likelihood.

We prove the contrapositive: assuming that an input passes with probability $> 1/4$, we will show that it is ϵ-close.

Let $X_1, \ldots, X_n$ be accepted with probability $> 1/4$. By equation (1), the number of bad indices is $< \epsilon n$. Therefore $\geq (1-\epsilon)n$ indices are good.

Claim 13 *If we delete all elements at bad indices, the remaining sequence is monotone.*

Proof Let $i < j$ be two good indices. Consider the paths in the binary-search tree from the root to i and to j. These two paths have some longest prefix common to both of them. It suffices to show that $x_i \le z \le x_j$.

There are two cases. If the path to x_i is a prefix of the path to x_j, then $x_i = z$ (they are the same node in the tree). Otherwise x_i is a descendant of a z's left or right child. Since i is good, then x_i must be a descendant of z's left child; for the same reason x_i must be smaller than z.

Therefore, $x_i \le z$ always. By symmetry, $z \le x_j$. Therefore $x_i \le x_j$. ■

The theorem follows from the claim. ■

Remark It is known that $\Omega\big((\log n)/\epsilon\big)$ samples is optimal.

2.3 Another example

Example 14: graph connectivity, attempt 1

Given A graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges having maximum degree at most d (we think of d as a large constant). The graph is represented as an adjacency list.

Goal Is the graph connected?

As before, answering this question with no error requires examining the entire graph: an example is the line graph L_n (i.e., a cycle with one edge removed). Therefore, we will have to compromise on the goal if we are limited to sublinear time.

Example 15: graph connectivity, attempt 2

Given A graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges having maximum degree at most d (we think of d as a large constant). The graph is represented as an adjacency list.

Goal Is the graph *close to connected*?

> We will say that a graph is ϵ-*close to connected* if it can be transformed into a connected graph by adding at most ϵdn edges.
>
> (An alternative definition exists, which allows adding or removing up to ϵdn edges, but on the other hand requires the resulting graph to still have maximum degree at most d. For simplicity we will use the addition-only definition given in the previous paragraph.)

Required behavior We require a 1-sided error:

- If the graph is connected, the test should pass with probability 1.
- If the graph is ϵ-far from connected, the test should fail with probability 3/4.

Proof Idea If a graph is ϵ-far from connected ⇒ it has many ($\ge \epsilon n$) connected components ⇒ many connected components are small ⇒ many nodes are in small connected components. ■

Algorithm

1. Choose $O(1/\epsilon d)$ nodes.
2. For each node s of these, run a BFS[2] (originating from it) until either:
   (a) $\geq 2/\epsilon d$ distinct nodes are discovered;
   (b) s is determined to belong to a connected component of size $\leq 2/\epsilon d$ nodes.
3. If 2b ever happens, reject G and halt.
4. Otherwise, accept.

Time complexity The number of loops is $O(1/\epsilon d)$. Each BFS costs up to $O(2/\epsilon d)$ steps. During the BFS, neighbor determination at each node is done by iterating its adjacency list, which can have length up to d. Therefore the time complexity is $O\left(\frac{1}{\epsilon d} \cdot \frac{2}{\epsilon d} \cdot d\right) = O(1/d\epsilon^2)$.

Lemma 16 *If G is ϵ-far from connected, then G has $\geq \epsilon dn$ connected components.*

Proof N connected components can be connected by adding $N-1$ edges. ■

Remark This proof is trickier if we use the alternative (max-degree-respecting) definition of ϵ-far.

Corollary 17 *If G is ϵ-far from connected, then it has $\geq \epsilon dn/2$ connected components of size less than $2/\epsilon d$.*

Proof Since G is ϵ-far, it has $L > \epsilon dn$ connected components. Let ℓ' be the number of connected components of size $< 2/\epsilon d$ and ℓ be the number of connected components of size $\geq 2/\epsilon d$. So $\ell + \ell' = L$.

Therefore, $\ell + \ell' \geq \epsilon dn$ (using the lemma).

Also $\ell \cdot \frac{2}{\epsilon d} \leq n$ (the LHS is the number of vertices in the connected components counted by ℓ), i.e., $\ell \leq \epsilon dn/2$.

The conclusion, $\ell' \geq \epsilon dn/2$, follows by combining the last two inequalities. ■

Corollary 18 *The fraction of nodes in V belonging to connected components smaller than $2/\epsilon d$ is at least $\epsilon d/2$.*

Proof For vertex $u \in V$, let $\mathcal{C}(u)$ denote the connected component of u and let $S(u)$ be the event that $|\mathcal{C}(u)| < 2/\epsilon d$. Then

$$\Prob_{u \in_R V}[S(u)] = \frac{|\{u \in V : S(u)\}|}{|V|} \geq \frac{\{\mathcal{C}(u) : u \in V \wedge S(u)\}}{|V|} \geq \frac{\epsilon dn/2}{n} = \frac{\epsilon d}{2}.$$

Intuitively, this bounds the number of nodes in small connected components from below by the number of such connected components. ■

Theorem 19 *The test passes connected graphs with certainty and fails ϵ-far graphs with probability at least $3/4$.*

Proof The first claim is obvious (step 2b will never occur). We show the second claim. We see that

$$\Prob[\mathsf{fail}] \geq 1 - \Prob[\mathsf{pass}] \geq 1 - \left(1 - \frac{\epsilon d}{2}\right)^{O(1/\epsilon d)} \geq 1 - e^{-c'} \geq \frac{3}{4}.$$

where the last inequality follows from choosing the constant c such that $e^{-c'} < 1/4$. ■

[2] breadth-first search