**CS 388R: Randomized Algorithms** Fall 2017

Lecture 14 — October 19, 2017

*Prof. Eric Price* *Scribes: Niels Kornerup, Aravind Gollakota*

**NOTE:** THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

# 1 Overview

In this lecture we finish our discussion of sampling and look at an efficient sampling-based algorithm for median-finding.

# 2 Sampling

Consider estimating the volume of a $d$-dimensional polytope given a way to tell if a point is inside it or not (i.e. a separation oracle).

- Let $p$ = probability a random point is in the polytope.
- Algorithm: Try $n$ random points, see what fraction of them are in the polytope.
- Estimate $\hat{p} = \frac{1}{n}\sum_{i=1}^{n} Z_i$, where $Z_i = 1$ if the $i^{\text{th}}$ point is in the polytope.

Question: how many samples do we need to get $\hat{p} \in (1 \pm \epsilon)p$?

- We get that $\mathbb{P}[\sum Z_i - \mathbb{E}[\sum Z_i] \geq \epsilon \cdot \mathbb{E}[\sum Z_i]] \leq 2e^{\frac{-\epsilon^2}{3}\sum Z_i}$ by a Chernoff bound, as the $Z_i$ are independent.
- This is equivalent to saying that $\mathbb{P}[|\hat{p} - p| \geq \epsilon p] \leq 2e^{\frac{-\epsilon^2 np}{3}}$
- Which then implies that we need $n = \frac{3}{p\epsilon^2} \times \log(\frac{2}{\delta})$ samples to get a $1 - \delta$ probability of $\hat{p} \in (1 \pm \epsilon)p$.
- But, this requires that we already know $p$...

What we really need is to make sure that $n \geq \frac{3}{p\epsilon^2}\log(\frac{2}{\delta})$, so we sample until we get $T$ hits.

- This will happen if $\sum_{i=1}^{n} Z_i \geq \frac{10}{\epsilon^2}\log(\frac{2}{\delta}) = T$
- Want $\hat{n} = (1 \pm \epsilon)T/p$

- For some $n'$, $\mathbb{P}[\sum_{i=1}^{n'} Z_i \notin (1 \pm \epsilon)n'p] \leq e^{-\frac{\epsilon^2}{3}n'}$
- Let us consider $n' = \frac{T}{1+\epsilon}$
  - $\mathbb{P}[\sum_{i=1}^{n'} Z_i \geq (1+\epsilon)n'p = Tp] \leq e^{-\frac{\epsilon^2}{3}n'p} = e^{\frac{-\epsilon^2 Tp}{1+\epsilon}} \leq e^{-\epsilon^2\frac{T}{4}}$
  - which is $\leq \delta/2$ if $T \geq \frac{4}{\epsilon^2}\log(\frac{2}{\delta})$
- Now, consider $n' = \frac{T}{1-\epsilon}$
  - $\mathbb{P}[\sum_{i=1}^{n'} Z_i \leq (1-\epsilon)n'p = Tp] \leq e^{-\frac{\epsilon^2}{2}n'p} \leq e^{-\epsilon^2\frac{T}{2}}$
  - which is $\leq \delta/2$ if $T \geq \frac{2}{\epsilon^2}\log(\frac{2}{\delta})$
- Thus, we get that if $T \geq \frac{4}{\epsilon^2}\log(\frac{2}{\delta})$ then with probability $1-\delta$ we get that $\frac{T}{(1+\epsilon)p} \leq \hat{n} \leq \frac{T}{(1-\epsilon)p}$, where $\hat{n}$ is the number of samples used
- This implies that $\frac{T}{\hat{n}} \in (1 \pm \epsilon)p$

To summarize, we've just shown that $T = O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ coin flips suffice to approximate $p$ to within $\epsilon$ with probability $\geq 1-\delta$.

# 3 Median-finding

Given an array of numbers, our task is to find the median. Here are some approaches that first come to mind:

**Quickselect.** This is a Quicksort derivative, and works as follows. Pick a random pivot, and separate the elements into those bigger and those smaller than the pivot. The median must lie in the bigger of the two segments, so recursively search within this segment. (Technically the median has a different rank within this segment, so we keep track of it; the implementation is really a FindElementOfRank($i$) rather than FindMedian.[1])

A fairly standard Quicksort-like analysis reveals that the expected number of operations in this algorithm is $O(n)$. What about high probability bounds? Unfortunately because we pick our pivot uniformly at random, it's not good. Specifically, the probability of our first $k$ choices all lying in the first $1/k$ fraction of the array is at least $1/k^k$. When this happens, after these $k$ steps our array has size $n(1-1/k)^k \approx n/e$. Thus there's a $1/k^k$ chance of taking $\Omega(kn)$ time (for all $k$), i.e. a $1/n$ chance of $\Omega(\frac{n\log n}{\log\log n})$ time. This is almost as bad as sorting, and tells us that high probability bounds definitely do not hold.

**Deterministic median-of-medians.** There is actually a deterministic algorithm that works. We divide our array into chunks of say 5, take the medians of each, and then take the median of these medians. We then use this as a pivot, again separating elements into two piles and recursing down the bigger one. It can be shown that this always only uses $O(n)$ comparisons.

[1] This applies to the rest of this lecture as well: our median-finding algorithms are really *selection* algorithms.

But from a practical point of view, the algorithm is not too simple to implement and also suffers from bad constants.

So we discard these approaches and frame as our goal an algorithm that is simple, randomized, has good high probability bounds, and has small constants.

Let's try to use sampling. Here is a first attempt at an algorithm:

- Choose a sample $S$ of size $s$ ($s \ll n/\log n$)
- Directly sort and take the median of $S$ (call it $m$), and use it as pivot
- Split the elements relative to the pivot, and recursively search for the median on the appropriate side

If $\ell$ is the number of elements on the median's side, then the running time for this algorithm may be expressed recursively as

$$T(n) = s \log s + n + T(\ell),$$

where the $s \log s$ comes from sorting $S$ and the $n$ from splitting the elements relative to the pivot. We can try to sample such that $m$ is close to the real median, i.e. that $\text{rank}(m) = (1 \pm \epsilon)n/2$ with high probability $(1 - 1/\text{poly}(n)$—although note that $n$ drops quickly as we recurse down, and this needs to dealt with carefully). Here we use $\text{rank}(m)$ to denote $m$'s position in the sorted array. If we could do this at all steps of the recursion, then we'd ensure that $\ell \leq (1 + \epsilon)n/2$ always and so

$$\begin{aligned} T(n) &\leq n + T(\frac{1+\epsilon}{2}n) \\ &= \frac{n}{1 - (1+\epsilon)/2} = \frac{2n}{1-\epsilon}. \end{aligned}$$

This isn't bad, and we could set about formalizing this. But instead we will do something better, something that is both simpler and has a better constant than 2.

The key idea is that instead of recursing down the entirety of one half, we can actually try to use more fine-grained information about our sample's median $m$: if it is really within $(1 \pm \epsilon)$ of the actual median, then we should really only be looking at $\epsilon n/2$ elements "on either side" of $m$. Formally, we will let $L$ be the $(1/2 - \epsilon)s^{\text{th}}$ largest element in $S$, and $H$ the $(1/2 + \epsilon)s^{\text{th}}$ largest. If $\text{rank}(L) \leq n/2$ and $\text{rank}(R) \geq n/2$, then the median will lie in $[L, H]$. We will see that we can make this happen with high probability. Notice that this idea is similar to our analysis of sampling.

Let's see what we can say about $\text{rank}(m)$. In fact, more generally consider $\text{rank}(x)$, where $x$ is the element of rank $\beta s$ in $S$ ($m$ corresponds to $\beta = 1/2$). We know that $\text{rank}(x) \leq k$ occurs iff at least $\beta s$ elements in $S$ have rank $\leq k$. The natural thing is to use Chernoff to bound this probability.

Let $X_i = 1$ if the $i^{\text{th}}$ sample has rank $\le k$, so that $\mathbb{E}[X_i] = k/n$. Then

$$\begin{aligned}\mathbb{P}[\text{rank}(x) \le k] &= \mathbb{P}[\sum_{i=1}^{s} X_i \ge \beta s] \\ &= \mathbb{P}\left[\sum_{i=1}^{s} X_i \ge \mathbb{E}[\sum_{i=1}^{s} X_i] + (\beta - \frac{k}{n})s\right] \\ &\le \exp(\frac{-2((\beta - \frac{k}{n})s)^2}{s}) = \exp(-2(\beta - \frac{k}{n})^2 s).\end{aligned}$$

To have this be $\le 1/\operatorname{poly}(n)$, it suffices to have $\frac{k}{n} = \beta - \sqrt{\frac{\log n}{s}}$. Thus whp (with high probability, i.e. $\ge 1 - 1/\operatorname{poly}(n)$), $\text{rank}(x) \ge n(\beta - \sqrt{\frac{\log n}{s}})$. Similarly (since we only used additive Chernoff) we can show that whp $\text{rank}(x) \le n(\beta + \sqrt{\frac{\log n}{s}})$. And so whp $\text{rank}(x) = (1 \pm \epsilon)\beta n$ if $s \ge \frac{1}{\epsilon^2}\log n$.

So now fix $\epsilon = \sqrt{\frac{\log n}{s}}$. Let $L$ be the $(1/2 - \epsilon)s^{\text{th}}$ largest element of $S$. Plugging $\beta = 1/2 - \epsilon$ into the above bound, we see that whp $\text{rank}(L) \in [n/2 - 2\epsilon n, n/2]$. Similarly if $R$ is the $(1/2 + \epsilon)s^{\text{th}}$ largest element of $S$, then whp $\text{rank}(R) \in [n/2, n/2 + 2\epsilon n]$. When both of these happen, we have that the true median is contained within $[L, H]$, an interval of length $\le 4\epsilon n = o(n)$, as desired.

To compute $[L, H]$, we need to make a linear pass over all elements to classify them as $\gtrless L, H$. If we compare to $L$ first and only to $R$ if necessary, then for roughly half the elements (specifically, $\ge n/2 - 2\epsilon n$ of them) we avoid comparing them to $R$ since they are less than $L$. So the number of comparisons here is $3n/2 + o(n)$.

Once we've found $[L, H]$, we can find the true median by simply sorting it. This takes at most $O(4\epsilon n \log(4\epsilon n))$ steps. Thus including the original step of sorting $S$ itself, the total number of operations is

$$O(s \log s) + \frac{3n}{2} + O(\epsilon n \log(\epsilon n)).$$

Plugging in $\epsilon = \sqrt{\log n/s}$, we see that this quantity is minimized at some appropriate $s$. Specifically, plugging in $s = n^{2/3}$ gives $3n/2 + n^{2/3}\log^{2/3} n = 3n/2 + o(n)$. Thus our new algorithm is indeed simpler, since it avoids recursion, and has a better constant (3/2 instead of 2).