COS 521: Advanced Algorithms*
Streaming Algorithms

1 Introduction

The emergence of the web and the explosion of digital data of various forms has created the need for very efficient algorithms to deal with large data sets. A search engine is a good example of an application that needs to efficiently process very large amounts of data. Think about processing query logs to compute statistical information about the queries (e.g. the number of distinct queries), or processing the collection of documents crawled by the search engine to identify and eliminate near duplicate documents. Both these applications can be readily solved if the data set is relatively small, for example, if it fits in main memory. However the sheer size of the data set renders most common approaches infeasible. In the case of computing the number of distinct queries, the obvious technique that comes to mind is to make one pass over the query log and record the set of distinct queries seen so far (e.g. by storing them in a hash table). This can then be used to report the number of distinct elements. A search engine like Google probably receives close to a billion queries every day. The hash table approach is not a very practical solutions in this setting. How would you ever hope to scale this solution to compute the number of distinct queries in a year ?

An emerging area in Computer Science is that of *streaming algorithms*, referring to very efficient algorithms that process very large amounts of data by making one or more passes over the data and recording very compact summaries, or *sketches* of the data. In constructing such compact summaries, we discard most of the data set, but retain some valuable information about it that still enables us to perform the computation we wanted. Ideally, we would like to be able to guarantee that such sketches allow us to compute the same things that could be computed with the entire data set. In most cases, this is too good to be true and we must settle for an approximation of what we wanted to compute. However, there is usually a tradeoff between the size of the sketch and the quality of the approximation it supports. In many cases of interest, by choosing our sketch sizes sufficiently large, we can ensure that the quantities of interest can be computed to any specified degree of accuracy. In other words, for any $\varepsilon > 0$, we can compute the quantities of interest to within a factor of $1 + \varepsilon$ of their correct values. Several ingenious, constructions of such sketches have been

* Princeton University, Spring 2011, Moses Charikar

devised that allow approximate computations to be performed using a very small summary of the data set. We will describe such sketch constructions for the problem of estimating the number of distinct queries in a query log and later, for estimating similarity of pairs of documents.

1.1 Estimating distinct elements

We would like to develop a procedure to estimate the number of distinct queries in the query log with very good accuracy. We will accomplish this by designing a *sketch* (a compact summary) that can be computed in a single pass over the data set. Further, we will be able to estimate the number of distinct queries from this sketch. By increasing the sketch size suitably, we will be able to get an approximation to the number of distinct elements to within any desired accuracy. Our sketch construction will be *randomized*, i.e. our algorithm will make some random choices initially and the sketch constructed will be a function of the random choices made. We will argue that the estimates produced from the sketch construction will very accurate with high probability.

The overall idea behind the sketch construction for estimating distinct elements is as follows: We first construct a random variable which is a function of the queries in the query log. This will be a quantity (a real number) that is quite simple and can be computed in a single pass over the entire data set by storing very little information. In fact we will design the random variable so that it is a function of the *set of distict queries* in the log, i.e. multiple copies of the same query will not affect its value. This is a random variable because its value will depend on some random choices made initially, as well as the set of distinct queries in the log. Further, we will show that the expected value of the random variable is a (monotone) function of the number of distinct queries. Knowing the expected value of this random variable exactly will allow us to compute the number of distinct queries exactly. Of course, we do not know how to compute the expectation of this random variable exactly. However, we will show that, by taking multiple independent copies of this random variable, we will be able to obtain a very good approximation of the expectation. This good approximation of the expected value will then allow us to get a good approximation of the number of distinct queries in the query log. Admittedly, this is somewhat indirect and it is completely unclear at this point how we are going to achieve this goal. Why should random variables have anything to do with the simple straightforward problem of computing the number of distinct queries ? Before you read the details, take a moment to review the overall approach again. Now take a deep breath and read on.

Let U be the universe of all queries. We will construct a random hash function $h : U \rightarrow [0, 1]$. This is a function that maps every query in the universe to a real number in the range $[0, 1]$. For now, let us make the (unrealistic) assumption that h is indeed a random hash function. This means two things. Firstly, $h(x)$ is a random variable that is uniformly distributed in $[0, 1]$. (Note that we say $h(x)$ is a random variable because its value depends on the initial random choice of hash function h.) Secondly, for distinct $x_1, \ldots x_n$, the values $h(x_1), \ldots, h(x_n)$ are mutually independent.

The algorithm initially picks such a random hash function h. We then applies $h()$ to

every query in the query log by making a single pass over the data. In doing so, we record the minimum value of $h(x)$ we have seen so far. Let Y be the minimum value of $h(x)$ over all queries in the query log. The random variable Y is the basis for the sketch construction for distinct elements.

First, lets establish some properties of this random variable Y. Observe that

1. Y can be computed in a single pass over the data set. Note that very little information about the data set needs to be stored in this process. In fact, we only need to store the minimum value of $h(x)$ over all queries x encoutered so far.

2. Y is a function of the collection of distinct queries in the query log. Multiple copies of a query do not affect the value of Y. (Verify this !)

What can we say about the expected value of Y ? It turns out that the expected value of Y is a function of the number of distinct queries in the query log. Suppose there are k distinct queries $x_1, \ldots, x_k$. Then, $h(x_1), \ldots, h(x_k)$ are k independent random numbers uniformly distributed in the range $[0, 1]$. Thus the distribution of random variable Y is the same as the distribution of the minimum of k independent random numbers chosen uniformly in $[0, 1]$. First lets get an intuitive understanding of what the expected value of Y ought to be. Roughly speaking, you expect k random numbers uniformly distributed in $[0, 1]$ to be roughly equally spread in the interval $[0, 1]$. Thus it reasonable to expect that the minimum of these k numbers is roughly $1/k$. In fact this is not very far from the truth. It turns out that $\mathbb{E}[Y] = \frac{1}{k+1}$ and further $\text{Var}[Y] \leq \frac{1}{(k+1)^2}$.

So we now have a random variable whose expectation is $1/(k+1)$. Recall that k the number of distinct elements is the quantity we set out to estimate. If we only knew the expectation of this random variable we constructed, we could derive k exactly from it. Well we can come close to achieving this goal. If we construct a collection of t independent copies $Y_1, \ldots, Y_t$ of the random variable Y, we could try to estimate the expectation by looking at the values of the t variables $Y_1, \ldots, Y_t$. First let's understand what it means to construct t independent random variables $Y_1, \ldots, Y_t$. Well we first pick t independent random hash functions $h_1, \ldots, h_t$ such that each of them map queries to $[0, 1]$. For the ith hash function h_i, let Y_i be the minimum value of $h_i(x)$ over all queries x in the log. The *sketch* consists of the t values $(Y_1, \ldots, Y_t)$. Note that this sketch can be computed in a single pass over the data set. The distribution of the random variables Y_i is exactly the same as the distribution of the random variable Y we analyzed before; thus its distribution is a function of the number k of distinct elements. However $Y_1, \ldots, Y_t$ are independent copies of this random variable, i.e. for a given value of k, the number of distinct elements, the variables $Y_1, \ldots, Y_t$ are mutually independent.

Recall the the whole point of defining $Y_1, \ldots, Y_t$ was to estimate the expectation of Y. Suppose we use the average value $\frac{Y_1 + \ldots + Y_t}{t}$ as an estimate of $E[Y]$. How large does t need to be so as to get an estimate within the range $(1 \pm \varepsilon)\mathbb{E}[Y]$? In order to do this we will apply

Chebyshev's inequality. Let $Z = \frac{Y_1+\ldots+Y_t}{t}$ Note that

$$\begin{aligned}
\mathbb{E}[Z] &= \mathbb{E}[Y] \\
\mathrm{Var}[Z] &= \frac{\mathrm{Var}[Y]}{t} \\
\Pr(|Z - \mathbb{E}[Z]| > \varepsilon\mathbb{E}[Z]) &\leq \frac{\mathrm{Var}[Z]}{(\varepsilon\mathbb{E}[Z])^2} \quad \text{(Chebyshev)} \\
&= \frac{\mathrm{Var}[Y]}{\varepsilon^2 t(\mathbb{E}[Y])^2} \leq \frac{1}{\varepsilon^2 t}
\end{aligned}$$

The last inequality follows since $\mathrm{Var}[Y] \leq (\mathbb{E}[Y])^2$. We say that we get a *good* estimate if our estimate is within a factor of $1 \pm \varepsilon$ of the correct value, otherwise we say that the estimate is *bad*. The above calculation shows that the probability that our estimate is bad is at most $\frac{1}{\varepsilon^2 t}$. Suppose we set $t = 4/\varepsilon^2$. Then the probability of a bad estimate is at most $1/4$.

Boosting the success probability

This is a pretty good scheme, but note that there is a fairly large probability $1/4$ of failure, i.e. of getting a bad estimate. How can we drive down the probability of failure to some specified (small) value δ ? One option is to take make t large enough such that the failure probability we get from Chebyshev's inequality is less than δ. If we do this, we need $t > \frac{4}{\varepsilon^2\delta}$. Note that t is the size of our sketch. Using this approach, we need a really large sketch size to drive the failure probability down to a small value. (think of $\delta = 1/10^6$ for example).

Here is a better approach. We construct s groups of $4/\varepsilon^2$ independent random variables each and compute the average value (mean) for each of these groups. Note that there are a total of $4s/\varepsilon^2$ independent copies of the random variable Y in this scheme. How do we combine the estimates obtained from different groups ? Here is the twist: we compute the *median* of the values computed by each group and use the median as our estimate.

Why is this a good scheme ? How large does s need to be to get a failure probability of δ ? Recall that, for any group, the probability that the estimate of that group is bad is at most $1/4$. How could the median of the s values be bad ? If the median of the s values is too low, then at least half of the s estimates of the groups must be too low. On the other hand, if the median of the s values is too high, then at least half of the s estimates of the groups must be too high. In either case, if the median is a bad estimator, then at least half of the estimates from the s groups must be bad. Now the expected number of bad estimates is at most $s/4$. By Chernoff bounds, the probability that the number of bad estimates is at least $s/2$ is $e^{-(2\ln 2-2+1)s/4} < e^{-s/11}$ This is an upper bound on the failure probability of our procedure. In order to get this failure probability to be less than δ, we set $s = 11\ln(1/\delta)$. This is a big win over the earlier proposal which required increasing the number of random variables (and hence the size of the sketch) by a factor of $1/\delta$. Again, think of a really low failure probability $\delta = 1/10^6$.

To recap, our overal scheme is to take $s = 11\ln(1/\delta)$ groups of $4/\varepsilon^2$ copies of the random variable Y. We compute the mean of the random variables in each group and compute the

median of the means obtained. The final value obtained is guaranted to be within $(1 \pm \varepsilon)\mathbb{E}[Y]$ with probability $1 - \delta$. This *median of means* scheme and its analysis occurs frequently and is useful to keep in mind.

Let's summarize what we accomplished. We constructed a random variable Y and by using a sketch of size $(44/\varepsilon^2)\log(1/\delta)$, we were able to estimate $\mathbb{E}[Y]$ with high accuracy and low failiure probability. Also $\mathbb{E}[Y] = 1/(k+1)$ where k is the number of distinct queries in the log. If A is the estimate we obtain for $\mathbb{E}[Y]$, we use $1/A$ as an estimate for k the number of distinct queries. Note that with high probability, A is guaranteed to be within $(1 \pm \varepsilon)$ of $\mathbb{E}[Y] = 1/(k+1)$. Hence with high probability, $1/A$ is guaranteed to be within the range $(1 \pm \varepsilon)(k+1)$.

This sounds pretty amazing! Compare this to the *brute force* method of storing a table with the set of distinct queries in it (space linear in the number of disitinct elements). This method actually requires only a *constant* amount of storage ! Of course, the constant depends on the quality of the approximation and the failure probability we are willing to tolerate, but it is independent of the actual number of distinct queries ! Who would have thought something like this was possible ?

But lets not get ahead of ourselves. We left a gaping hole in our scheme by making the big assumption up front that the hash function $h : U \to [0, 1]$ was a random hash function. Of course, we cannot really pick a random hash function, so we need to replace this with something else. (Think about why we cannot really pick a truly random hash function where the hash values are mutually independent – how would you represent such a hash function? What's wrong with using a hash function based on a pseudorandom number generator such as rand or drand48 in C or random in java?)

Fortunately, it turns out that we can replace the random hash function by a simple pairwise independent hash function of the form $h(x) = \frac{(ax+b \mod p)}{p}$. The process of producing an estimate using this hash family differs slightly from what we did here assuming completely random hash functions (means are replaced by medians etc), but the basic approach is similar. It is not too difficult to analyze this estimator and prove that it produces an estimate in the range $[k/3, 3k]$ with high probability if k is the number of distinct queries. It turns out that one can also get estimates that are good to within any degree of accuracy by modifying the estimator somewhat, but we will not explore this here.