Using only memory equivalent to 5 lines of printed text, you can estimate with a
typical accuracy of 5 per cent and in a single pass the total vocabulary of Shake-
speare. This wonderfully simple algorithm has applications in data mining, esti-
mitating characteristics of huge data flows in routers, etc. It can be implemented
by a novice, can be fully parallelized with optimal speed-up and only need minimal
hardware requirements. Theres even a bit of math in the middle!


As we saw in Lecture 1, hashing can be thought of as a way to rename an address space.
For instance, a router at the internet backbone may wish to have a searchable database of
destination IP addresses of packets that are whizzing by. An IP address is 128 bits, so the
number of possible IP addresses is $2^{128}$, which is too large to let us have a table indexed
by IP addresses. Hashing allows us to rename each IP address by fewer bits. In Lecture 1
this hash was a number in a finite field (integers modulo a prime $p$). In recent years large
data algorithms have used hashing in interesting ways where the hash is viewed as a real
number. For instance, we may hash IP addresses to real numbers in the unit interval $[0,1]$.

Example 1 (Dartthrowing method of estimating areas) Suppose gives you a piece
of paper of irregular shape and you wish to determine its area. You can do so by pinning
it on a piece of graph paper. Say, it lies completely inside the unit square. Then throw a
dart $n$ times on the unit square and observe the fraction of times it falls on the irregularly
shaped paper. This fraction is an estimator for the area of the paper.

Of course, the digital analog of throwing a dart $n$ times on the unit square is to take a
random hash function from $\{1, \ldots, n\}$ to $[0,1] \times [0,1]$.

Strictly speaking, one cannot hash to a real number since computers lack infinite preci-
sion. Instead, one hashes to rational numbers in $[0,1]$. For instance, hash IP addresses to
the set $[p]$ as before, and then think of number “$i \mod p$” as the rational number $i/p$. This
works OK so long as our method doesn’t use too many bits of precision in the real-valued
hash.

A general note about sampling. As pointed out in Lecture 3 using the random variable
"Number of ears," the expectation of a random variable may never be attained at any point
in the probability space. But if we draw a random sample, then we know by Chebysev’s
inequality that the sample has chance at least $1 - 1/k^2$ of taking a value in the interval
$[\mu - k\sigma, \mu + k\sigma]$ where $\mu, \sigma$ denote the mean and variance respectively. Thus to get any
reasonable idea of $\mu$ we need $\sigma$ to be less than $\mu$. But if we take $t$ independent samples
(even pairwise independent will do) then the variance of the mean of these samples is $\sigma^2/t$.
Hence by increasing $t$ we can get a better estimate of $\mu$. 

1
1 Estimating the cardinality of a set that’s too large to store

Continuing with the router example, suppose the router wishes to maintain a count of the
number of distinct IP addresses seen in the past hour. It would be too wasteful to actually
store all the IP addresses; an approximate count is fine. This is also the application alluded
to in the quote at the start of the lecture.

An idea: Pick $k$ random hash functions $h_1, h_2, \ldots, h_k$ that map a 128-bit address to
a random real number in $[0,1]$. (For now let’s suppose that these are actually random
functions.) Now maintain $k$ registers, initialized to 0. Whenever a packet whizzes by, and
its IP address is $x$, compute $h_i(x)$ for each $i$. If $h_i(x)$ is less than the number currently
stored in the $i$th register, then write $h_i(x)$ in the $i$th register.

Let $Y_i$ be the random variable denoting the contents of the $i$th register at the end. (It
is a random variable because the hash function was chosen randomly. The packet addresses
are not random.) Realize that $Y_i$ is nothing but the lowest value of $h_i(x)$ among all IP
addresses seen so far.

Suppose the number of distinct IP addresses seen is $N$. This is what we are trying to
estimate.

Fact: $E[Y_i] = \frac{1}{N+1}$ and the variance of $Y_i$ is $1/(N+1)^2$.

The expectation looks intuitively about right: the minimum of $N$ random elements in
$[0,1]$ should be around $1/N$.

Let’s do the expectation calculation. The probability that $Y_i$ is $z$ is the probability that
one of the IP addresses mapped to $z$ and all the others mapped to numbers greater than $z$.

$$E[Y_i] = \int_{z=0}^{1} \Pr[Y_i > z] dz = \int_{z=0}^{1} (1 - z)^N dz = \frac{1}{N+1}.$$ 

(Here’s a slick alternative proof of the $1/(N+1)$ calculation. Imagine picking $N+1$
random numbers in $[0,1]$ and consider the chance that the $N+1$th element is the smallest.
By symmetry this chance is $1/(N+1)$. But this chance is exactly the expected value of the
minimum of the first $N$ numbers. QED.)

Since we picked $k$ random hash functions, the $Y_i$’s are iid. Let $\overline{Y}$ be their mean. Then
the variance of $\overline{Y}$ is $1/k(N+1)^2$, in other words, $k$ times lower than the variance of each
individual $Y_i$. Thus if $1/k$ is less than $\epsilon^2$ the standard deviation is less than $\epsilon/(N+1)$,
whereas the mean is $1/(N+1)$. Thus with constant probability the estimate $1/\overline{Y}$ is within
$(1 + \epsilon)$ factor of $N$.

All this assumed that the hash functions are random functions from 128-bit numbers to
$[0,1]$. Let’s now show that it suffices to pick hash functions from a pairwise independent
family, albeit now yielding an estimate that is only correct up to some constant factor.
Specifically, the algorithm will take $k$ pairwise independent hashes and see if the majority
of the min values are contained in some interval of the type $[1/3x, 3/x]$. Then $x$ is our
estimate for $N$, the number of elements. This estimate will be correct up to a factor 3 with
probability at least $1 - 1/k$.

What is the probability that we hash $N$ different elements using such a hash function
and the smallest element is less than $1/3N$? For each element $x$, $\Pr[h(x) < 1/3N]$ is at
most $1/3N$, so by the union bound, the probability in question is at most $N \times 1/3N = 1/3$. 
Similarly, the probability that \( \Pr[\exists x : h(x) \leq 1/N] \) can be lower bounded by the inclusion-exclusion bound.

**Lemma 1 (inclusion-exclusion bound)**

\[
\Pr[A_1 \lor A_2 \ldots \lor A_n], \text{ the probability that at least one of the events } A_1, A_2, \ldots, A_n \text{ happens, satisfies}
\]
\[
\sum_i \Pr[A_i] - \sum_{i \neq j} \Pr[A_i \land A_j] \leq \Pr[A_1 \lor A_2 \ldots \lor A_n] \leq \sum_i \Pr[A_i].
\]

Since our events are pairwise independent we obtain

\[
\Pr[\exists x : h(x) \leq 1/N] \geq N \times \frac{1}{N} - \left( \begin{array}{c} N \end{array} \right) \frac{1}{N^2} \geq \frac{1}{2}.
\]

Using a little more work it can be shown that with probability at least 0.6 the minimum hash is in the interval \([1/3N, 3/N]\). (NB: These calculations can be improved if the hash is from a 4-wise independent family.) Thus if we repeat with \( k \) hashes, the probability that the majority of min values are not contained in \([1/3N, 3/N]\) drops as \( O(1/k) \).

## 2 Estimating document similarity

One of the aspects of the data deluge on the web is that often one finds duplicate copies of the same thing. Sometimes the copies may not be exactly identical: for example mirrored copies of the same page but some are out of date. The same news article or blog post may be reposted many times, sometimes with editorial comments. By detecting duplicates and near-duplicates internet companies can often save on storage by an order of magnitude.

We present a technique called *similarity hashing* that allows this approximately. It is a hashing method such that the hash preserves some "sketch" of the document. Two documents' similarity can be estimate by comparing their hashes. This is an example of a burgeoning research area of hashing while preserving some *semantic* information. In general finding similar items in databases is a big part of data mining (find customers with similar purchasing habits, similar tastes, etc.). Today's simple hash is merely a way to dip our toes in these waters.

So think of a document as a set: the set of words appearing in it. The *Jaccard similarity* of documents/sets \( A, B \) is defined to be \( |A \cap B| / |A \cup B| \). This is 1 iff \( A = B \) and 0 iff the sets are disjoint.

Basic idea: Pick a random hash function mapping the underlying universe of elements to \([0, 1]\). Define the hash of a set \( A \) to be the *minimum* of \( h(x) \) over all \( x \in A \). Then by symmetry, \( \Pr[\text{hash}(A) = \text{hash}(B)] \) is exactly the Jaccard similarity. (Note that if two elements \( x, y \) are different then \( \Pr[h(x) = h(y)] = 0 \) when the hash is real-valued. Thus the only possibility of a collision arises from elements in the intersection of \( A, B \).) Thus one could pick \( k \) random hash functions and take the fraction of instances of \( \text{hash}(A) = \text{hash}(B) \) as an estimate of the Jaccard similarity. This has the right expectation but we need to repeat with \( k \) different hash functions to get a better estimate.

The analysis goes as follows. Suppose we are interested in flagging pairs of documents whose Jaccard-similarity is at least 0.9. Then we compute \( k \) hashes and flag the pair if at least \( 0.9 - \epsilon \) fraction of the hashes collide. Chernoff bounds imply that if \( k = \Omega(1/\epsilon^2) \) this
flags all document pairs that have similarity at least 0.9 and does not flag any pairs with similarity less than $0.9 - 3\epsilon$.

To make this method more realistic we need to replace the idealized random hash function with a real one and analyse it. That is beyond the scope of this lecture. Indyk showed that it suffices to use a $k$-wise independent hash function for $k = \Omega(\log(1/\epsilon))$ to let us estimate Jaccard-similarity up to error $\epsilon$. Thorup recently showed how to do the estimation with pairwise independent functions. This analysis seems rather sophisticated; let me know if you happen to figure it out.

Bibliography


