Randomized Algorithms 2016/7
Lecture 3

Ball and bins, Martingales, Concentration bounds and Bloom Filters*

Moni Naor

1 Simultaneous Message Model and Sketching

The simultaneous message model for evaluating a function $f(x, y)$: Alice and Bob share a random string. They receive inputs x and y respectively and each should send a message to a referee, Charlie, who should evaluate the function $f(x, y)$. They may also have their own private source of randomness. The goal is for Alice and Bob to send short messages to Charlie.

For the equality function (as well as most interesting functions) we must have a probability of error or else Alice and Bob send the full inputs. If the common random string encodes a pair-wise independent hash function then we are in good shape.

Another view of this model is as sketching. Alice and Bob and any other participant publish a sketch of their input that allows computing functions on it.

Question: what can be done in the Simultaneous Message Model when there is **no public randomness**, specifically for the equality function? Answer, there is a $O(\sqrt{n})$ algorithm. Hint: think of good error correcting code.

2 Martingales

We reviewed Martingales: that we are dealing with a sequence of random variables $Z_0, Z_1, \ldots, Z_m$ so that for all $0 \leq i < m$ we have

$$E[Z_{i+1}|Z_i] = Z_i \tag{1}$$

The name 'martingale' comes from the betting world. The typical story is that Z_i represents the wealth after a sequence of fair bets where the winnings in the ith round are represented by Y_i with $E[Y_i] = 0$.

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

One issue that came up in the discussion is that in the literature one often sees the requirement (1) phrased instead as

$$E[Z_{i+1}|Z_i, Z_{i-1}, \ldots, Z_0] = Z_i \tag{2}$$

The question is whether these two requirements are equivalent and whether we can get the nice concentration for both of them. They are *not* equivalent as the following example shows. Consider a random walk on the integers with a reflecting wall at 0 where the first step chooses whether the walk is in the non negative or non positive integers. That is $Z_0 = 0$, the first step is random in $\{-1, 1\}$ and all subsequent steps are random in $\{-1, 1\}$ if $Z_i \neq 0$ and reflect to the previous step if $Z_i = 0$ (i.e. $Z_{i+1} = Z_{i-1}$).

Now the sequence is not a martingale according to Definition (2), since given $Z_i, Z_{i-1}, \ldots, Z_0$, if $Z_i = 0$ then we know that $Z_{i+1} = Z_{i-1}$. But Definition (1) holds, since given $Z_i = 0$ we have no idea whether we arrived from the positives or negatives.

Under both definitions we have Azuma's inequality:

Theorem 1. *Let $c = Z_0, Z_1, \ldots Z_m$ be a martingale such that $|Z_{i+1} - Z_i| \leq 1$ for all $0 \leq i < m$. Then*

$$\Pr[|Z_m - c| > \lambda\sqrt{m}] < 2e^{-\lambda^2/2}$$

The proof can be found in Alon Spencer [1].

A *Doob Martingale* is one obtained when $Z_i = E[f(W_1, W_2, \ldots W_n)|W_1, W_2, \ldots, W_i]$ where the W_i's are random variables in some set A and $f : A^n \mapsto R$. This is very convenient when trying to show that the performance of an algorithm is close to its expected value with high probability. An example we will discuss is the number of '0's in a Bloom filter with truly random hash functions and similarly throwing n balls into n bins at random, what can you say about the expected number of vacant bins and how concentrated is this value around the expectation?

3 Bloom Filters

A Bloom filter is a data structure that represents a set $S \subseteq U$ of size n *approximately* in the following sense: for every $x \in S$ it always answers 'yes' and for $x \notin S$ it answers 'yes' with probability at most ϵ. The probability is over the randomness used for generating the representation. Bloom filters are named after Burton Bloom who suggested them in 1970 [3]. It is one the most useful data structures (See the survey [4].

Representing a set precisely takes $\lceil \log \binom{u}{n} \rceil$ bits at least, since this is log the number of different subset of size n and also there exists a representation that uses this many bits. A good approximation for $\log \binom{u}{n}$ is $n \log(u/n)$ where we are loosing at most an $O(n)$ additive factor which we can get using the following:

$$(n/e)^n < n! < e\sqrt{n}(n/e)^n.$$

How much can we save by using an approximate representation? If the representation takes m bits at most, then for any S there exists a representation $W \in \{0, 1\}^m$ with at most $\epsilon(u - n)$

false positive (this true is since there is always a point that achieves at most the expected false positive rate. To get from W an exact representation of S we need to store a set of size n out of the false positives under W plus the 'true' positives, namely S, which can be done using at most $\lceil \log \binom{\epsilon(u-n)+n}{n} \rceil$ bits. So we get that

$$\left\lceil \log \binom{\epsilon(u-n)+n}{n} \right\rceil + m \geq \left\lceil \log \binom{u}{n} \right\rceil .$$

Therefore m has to be at least $n \log(1/\epsilon) - O(n)$.

The 'abstract' construction we was to hash S to a range of size n/ϵ and then solve the exact dictionary problem using an optimal number of bits (we did not talk how to achieve that constructively). If the initial hash uses a pairwise independent function g then the probability of an element $x \notin S$ colliding with any element is S is bound by $n \cdot \epsilon/n = \epsilon$. The number of bits required is thus —g— (which is $2 \log u$ plus $n \log(1/\epsilon)$. So this result is very tight.

The original and common way of implementing Bloom filters is different and uses a $\{0, 1\}$ vector.

4 Hash Tables

Hash tables is very well studied subject in computer science and one of the more useful practices. Knuth's "The Art of Computer Programming" Volume 3 devotes a lot of space to the various possibilities and the origin of the idea is attributed to a 1956 paper by Arnold Dumey [7]. A major issue is how to resolve collisions and popular suggestions are chaining and Open addressing (or closed hashing). For the latter we need to specify a probing scheme and one of the more popular ones is linear probing which takes advantage of the locality properties of computer memory (in the various levels of hierarchy).

The 'modern' era of investigating hashing can be seen in the work of Carter and Wegman [5] who suggested the idea of thinking of the input as being worst case and the performance is investigated when the hash function is chosen at random from a predefined family (rather than assuming a truly random function).

The simplest way to obtain dictionaries with expected $O(1)$ per operation is to use chained hashing with a table of size $O(n)$. Here, it is enough to choose the hash function from a δ-universal family, for δ which is $O(1/n)$. The *expected length* of a chain is now $O(1)$ and the length of the chain is what determines the cost of an operation. Note however that there will be long chains. Universality on its own only suffices to guarantee that the expected length of the *longest* chain is $O(\sqrt{(}n))$. The upper bound follows from considering all potential collisions: there are $\binom{n}{2}$ of them. Each collision occurs with probability $O(1/n)$, so the expected number of collisions is $O(n)$. On the other hand, in a chain of length ℓ there are $\binom{\ell}{2}$ collisions. Therefore the expected length of the longest chain cannot be larger than $O(\sqrt{(}n))$. For the lower bound see Alon et al. [2].

What happens if the hash function is truly random? Then this is the "ball and bins" scenario which we will talk in the next lecture and here the heaviest bin/chain is likely to contain $\Theta(\log n / \log \log n)$ elements.

Universality on its own also just guarantees *expected* $O(1)$ performance and not, say, high probability

$(1 - 1/poly(n))$ amortized $O(1)$ performance. There are several ways to obtain this sort of result. In future lectures we will explore "Cuckoo Hashing" a method that uses two hash functions h_1 and h_2 and where each element x in the set resides either in location $h_1(x)$ or location $h_2(x)$. This means that lookup requires just two accesses to the memory. Insertion may be more involved and requires relocating elements.

A lecture by Eric Demaine on hashing is available and recommended [6].

References

[1] Noga Alon, Joel H. Spencer, **The Probabilistic Method**, Wiley, 1992.

[2] Noga Alon, Martin Dietzfelbinger, Peter B. Miltersen, Erez Petrank, and Gabor Tardos, *Linear Hashing* Journal of the ACM, Vol. 46(5), 1999. `http://www.brics.dk/RS/97/16/BRICS-RS-97-16.pdf`

[3] Burton Bloom. *Space/Time Tradeoffs in Hash Coding with Allowable Errors*, Communications of the ACM 13:7 (1970), 422426.

[4] A Broder and M Mitzenmacher, *Network Applications of Bloom Filters: A Survey*, Internet Mathematics, 2002

[5] J. L. Carter and M. N. Wegman, Universal classes of hash functions, J. Comput. Syst. Sci. 18 (1979) 143–154.
`http://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/universalclasses.pdf`

[6] Eric Demaine, Lecture 10 in course 6.851: Advanced Data Structures (Spring'12) on Dictionaries, `https://courses.csail.mit.edu/6.851/spring12/lectures/L10.html`

[7] Arnold Isaac Dumey, *Indexing for rapid random-access memory*, Computers and Automation 5 (12), 6–9, 1956.