Randomized Algorithms 2020-1
Lecture 12

Bloom Filter, Derandomiaztion, Cuckoo Hashing *

Moni Naor

1 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 [5]. It is one the most useful data structures (See the survey [7].

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 saw 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, but one method is given here [3]). 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 $n \cdot \epsilon/n = \epsilon$.

*These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. In the interest of brevity, most references and credits were omitted.

The number of bits required is thus —g— (which is $2 \log u$ plus $n \log(1/\epsilon)$. So this result is very tight.

We saw a construction based on Cuckoo Hashing where instead of storing the element x itself we store $g(x)$ for a pair-wise independent hash function g.

The original and common way of implementing Bloom filters is different and uses a $\{0,1\}$ vector. You can read about that in Chapter 5 of Mitzenmacher-Upfal.

2 Random Graphs

There are several models for random graphs. Among the more significant ones are Erdős–Rényi models

- $G(n,p)$: n nodes, each edge exists with probability p independently of the others.
- $G(n,M)$: n nodes, the graph is chosen uniformly at random from the collection of all graphs which have n nodes and M edges.

The expected number of edges in $G(n,p)$ is $\binom{n}{2}p$. Therefore, as a rule of thumb, $G(n,p)$ behaves similarly to $G(n,M)$ with $M = \binom{n}{2}p$.

You can rad a lot about properties of random graphs and algorithms in both Alon-Spencer and Mitzenmacher-Upfal.

The questions relevant to Cuckoo Hashing are What happens with very sparse graphs, i.e. $np < 1$.

3 Derandomziation

How to construct random looking hash functions? One aproach is throug k-wise Independence. Definition: A family H of hash functions $h_i : U \mapsto [0, ..., m-1]$ is k-wise independent if for any k distinct $x_1, \ldots, x_k \in U$ and for any $y_1, ..., y_k \in [0, ..., m-1]$:

$$\Pr_{h \in H}[h(x_1) = y)1^h(x_2) = y_2..^h(x_k) = y_k] = 1/m^k$$

The 'classical' construction of k-wise Independence is using random polynomials of degree $k-1$. The down side of this construction is that it take time k to evaluate at a given point. There are approximate constructions of k-wise Independence that are based on cuckoo-hashing, where evaluation is much faster. See [9**?** , 4].

In general, the compression argument we gave works well with $\log n$-wise Independence. We need this much Independence to argue that there are no large components. It is sufficient, since we will never consider more than $\log n$ long chains.

We would like to get conditions where we can use k-wise independence instead of full independence for k which is polylog.

We can appeal to a result of Braverman [6] that says that poylog-wise independence fools all of $AC0$. We need to define 'fools' and $AC0$.

AC^0 is the class of function $\{0,1\}^n \mapsto \{0,1\}$ that are computable by circuits that are poly size, constant depth, unbounded fan-in gates of 'And's and 'Or's. Examples for functions *not* in AC^0 are the parity function or those involving exact counting. Approximate counting is in AC^0.

By a distribution μ fooling a complexity class we essentially mean that it is indistinguishable for a function in the complexity class whether it gets a uniformly at random input or one drawn from μ Fools means that the the probability of outputting a '1' is close on both distributions.

Theorem: Let C be circuit of depth d and size m. Let μ be an r-wise independent distribution for $r \geq cd(logm)^{O(d^2)}$. Then C cannot tell μ from the uniform distribution.

Note that for $m = n^{O(logn)}$, r is polylog(n). The way to apply the theorem in the context of data structures is to argue that checking whether the data structure works properly of not can be done with a low depth circuit (in the case of cuckoo hashing, it can be done with a depth 3 circuit of size $n^{O(logn)}$). See Arbirman, Naor and Segev [2].

References

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

[2] Yuriy Arbitman, Moni Naor, Gil Segev, *De-amortized Cuckoo Hashing: Provable Worst-Case Performance and Experimental Results.* ICALP 2009: 107-118.

[3] Yuriy Arbitman, Moni Naor, Gil Segev: Backyard Cuckoo Hashing: Constant Worst-Case Operations with a Succinct Representation. FOCS 2010: 787-796

[4] M. Aumuller, M. Dietzfelbinger and P. Woelfel. *Explicit and efficient hash families suffice for-cuckoo hashing with a stash.* Algorithmica, 70(3):428–456, 2014

[5] Burton Bloom. *Space/Time Tradeoffs in Hash Coding with Allowable Errors*, Communications of the ACM 13:7 (1970), 422—426.

[6] Mark Braverman: Polylogarithmic independence fools AC0 circuits. J. ACM 57(5): 28:1-28:10 (2010).

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

[8] 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`

[9] M. Dietzfelbinger and P. Woelfel. *Almost random graphs with simple hash functions.* 35th ACM Symposium on Theory of Computing (STOC), pages 629–638, 2003.

[10] A. Pagh and R. Pagh. *Uniform hashing in constant time and optimal space.* SIAM Journal on Computing, 38(1):85–96, 2008