CS271 Randomness & Computation **Spring 2018**

Lecture 10: February 15

Instructor: Alistair Sinclair

10.1 Pairwise independent random variables

Definition 10.1 *A collection $\{X_i\}_{i=1}^n$ of (discrete) random variables over the same probability space is said to be* pairwise independent *if, for all $i \neq j$ and all possible pairs of values a, b,*

$$\Pr[X_i = a \wedge X_j = b] = \Pr[X_i = a] \Pr[X_j = b].$$

*More generally, the collection is k-*wise independent *if, for every subset $I \subseteq \{1, \ldots, n\}$ with $|I| \leq k$, and every set of values $\{a_i\}$,*

$$\Pr[\bigwedge_{i \in I} X_i = a_i] = \prod_{i \in I} \Pr[X_i = a_i].$$

Remark 10.2 *k-*wise independent *variables may be equivalently defined as the set $\{X_i\}_{i=1}^n$ of random variables over the same probability space such that for every subset $I \subseteq \{1, \ldots, n\}$ with $|I| < k$ and every set of values $\{a_i\}$ and $j \notin I$,*

$$\Pr[X_j = b | \bigwedge_{i \in I} X_i = a_i] = \Pr[X_j = b]$$

k-wise independence is of course a much weaker requirement than mutual independence. However, as we shall see in this lecture, in certain applications k-wise independent (or even pairwise independent) families behave as well as mutually independent families as far as the application is concerned. The major advantage is that k-wise independent families can often be constructed with less randomness (and hence also represented more compactly) than independent ones. We shall see several examples in this lecture.

10.1.1 Probability amplification using pairwise independence

Let $L \subseteq \{0,1\}^n$ be a language and let A be a one-sided error Monte-Carlo algorithm for L, i.e.
$x \in L \Rightarrow \Pr[A \text{ outputs yes}] \geq 1/2$
$x \notin L \Rightarrow \Pr[A \text{ outputs yes}] = 0$

Using t independent runs of A we achieve error probability $\leq 2^{-t}$. Assuming that A requires m random bits, it follows that in order to achieve error probability $1/r$ we need $m \log r$ random bits by this method. However, randomness is an expensive resource, so it is natural to ask if there is another way to amplify the error probability using less randomness. The following construction shows, perhaps surprisingly, that this is indeed possible.

Theorem 10.3 **[Chor/Goldreich [CG89]]** *For any $r \leq 2^m$ we can achieve error probability $\leq 1/r$ using only $2m$ random bits (and with running time $O(rm)$).*

Proof: Note that the possible executions of A can be identified with strings in $\{0,1\}^m$ corresponding to the coin flips used by A. Pick $r < 2^m$ *pairwise* independent uniform samples from $\{0,1\}^m$. Let X_i be the outcome of the algorithm on the i^{th} sample, i.e., let

$$X_i = \begin{cases} 1 & \text{if } A \text{ outputs yes on this coin-flipping sequence;} \\ 0 & \text{otherwise.} \end{cases}$$

Define $X = \sum_{i=1}^{r} X_i$.

By the usual argument, using Chebyshev's inequality:

$$\Pr[X = 0] \leq \Pr[|X - \mathrm{E}X| \geq \mathrm{E}X] \leq \frac{\mathrm{Var}(X)}{(\mathrm{E}X)^2}.$$

Calculating the expression for the variance, we get

$$\mathrm{Var}(X) = \sum_i \mathrm{Var}(X_i) + \sum_{i \neq j} \mathrm{Cov}(X_i, X_j)$$

By definition, the covariance between any two pairwise independent random variables is 0, so in the above expression $\mathrm{Var}(X) = \sum \mathrm{Var}(X_i)$. For our 0-1 valued random variables X_i we have $\Pr[X_i = 1] = p \geq 1/2$, and $\mathrm{Var}[X_i] = p(1-p)$. Therefore $\mathrm{Var}[X] = rp(1-p)$, and we have

$$\frac{\mathrm{Var}(X)}{(\mathrm{E}X)^2} = \frac{r \cdot p(1-p)}{(rp)^2} = \frac{1-p}{p} \times \frac{1}{r} \leq \frac{1}{r},$$

since $p \geq 1/2$.

We observe that we have achieved a linear reduction in the error probability by using pairwise independent samples (instead of mutually independent ones). It remains now to show how these pairwise independent samples can be generated using only $2m$ random bits, which is a substantial saving over the $m \log r$ random bits we would need to achieve the same error reduction using mutually independent trials.

An example of such a construction is the following:

Assume q is a prime number, $2^m < q < 2^{m+1}$. (Such a prime q always exists. Actually, all we need is that $\mathbb{Z}_q$ is a field, so q could be any prime power). Pick a, b uniformly at random from $\mathbb{Z}_q$. Define $f_{a,b} : \mathbb{Z}_q \to \mathbb{Z}_q$ by

$$f_{a,b}(x) = ax + b.$$

Claim 10.4 *The random variables $\{f_{a,b}(x) : x \in \mathbb{Z}_q\}$ are pairwise independent and uniform over $\mathbb{Z}_q$*

Proof: It is a simple exercise to check that, for all $x, z \in \mathbb{Z}_q$, we have

$$\Pr_{a,b}[f_{a,b}(x) = z] = \Pr_{a,b}[ax + b = z] = 1/q,$$

so the r.v.'s are uniform. Also, for $x \neq y$, in order to have $f_{a,b}(x) = z_1$ and $f_{a,b}(y) = z_2$ the following system of linear equations must be satisfied:

$$\begin{aligned} ax + b &= z_1; \\ ay + b &= z_2. \end{aligned}$$

This system has determinant

$$\begin{vmatrix} x & 1 \\ y & 1 \end{vmatrix}$$

which is non-zero for $x \neq y$. Therefore the system has a unique solution for a and b, which implies that

$$\Pr_{a,b}[f_{a,b}(x) = z_1 \wedge f_{a,b}(y) = z_2] = 1/q^2.$$

This proves the claim. ■

The theorem now follows once we observe that the number of random bits used to generate the r.v.'s in the Claim is only $2m$. ■

Note: A detail in the above construction is that the random variables take values in $\mathbb{Z}_q$, rather than in $\{0, 1, \ldots, 2^m - 1\}$ as required in the application. A simple fix for this is to just discard values that are larger than $2^m - 1$. Since, we are discarding only a small fraction (at most half) of the values, in expected $O(m)$ bits, we can generate a set of pairwise independent random values in $\mathbb{Z}_q$ such that each of them is less than $2^m - 1$.

There is a wealth of literature on saving random bits in randomized algorithms (a topic generally referred to as "randomness amplification"). The above technique, sometimes known as "two-point sampling," is due in this context to Chor and Goldreich [CG89], though the essential idea dates back to Joffe [J74]. A more powerful technique based on random walks on expanders was pioneered by Ajtai/Komlós/Szemerédi [AKS87], who showed that it is possible to use only $m + O(r)$ random bits while reducing the error probability to $2^{-\Omega(r)}$. The state of the art is due to Zuckerman [Z96], who was able to achieve error probability 2^{-r} using only $(1 + \epsilon)(m + r)$ random bits, for any $\epsilon > 0$, using extractors.

Before we move on, here is another construction of pairwise independent random variables taking values in $\{0, 1\}^n$ (which may in some instances be more useful than the family in Claim 10.4 above which takes values in $\mathbb{Z}_q$). Let A be a random $n \times m$ *Toeplitz matrix*. Such a matrix is constructed by picking the entries of the first row and column uniformly at random from $\{0, 1\}$, and then copying each value all along its corresponding diagonal. In addition, let b be a random $\{0, 1\}$ vector of length n. Define $h_{A,b}(x) = Ax + b$, for $x \in \{0, 1\}^m$. It is left as an **exercise** to check that the family $\{h_{A,b}(x) : x \in \{0, 1\}^m\}$ consists of 2^m pairwise independent uniform random variables over $\{0, 1\}^n$. Note that the number of random bits used is only $2n + m$.

10.1.2 Derandomization using k-wise independence

We will consider the problem of coloring the edges of the complete graph K_n with two colors. We want to find a 2-coloring of the graph that avoids monochromatic k-cliques. As we saw in Lecture 5, coloring each edge red or blue with probability 1/2 independently yields an expected number of monochromatic k-cliques

$$\mathrm{E}[X] = \binom{n}{k} 2^{-\binom{k}{2}+1} =: f(k).$$

Hence there exists a coloring with at most $f(k)$ monochromatic k-cliques. (In Lecture 5 we applied this observation to argue that for $n = 2^{k/2}$ we get $f(k) < 1$, so there must exist a coloring with no monochromatic k-cliques.)

The above algorithm can be derandomized using the method of conditional probabilities (Lecture 6), simply by coloring one edge at a time and following the path with the least expected value of X. [**Exercise:** Verify that this can indeed be done in time $O(n^k)$; the main task is to keep track of $\mathrm{E}[X]$ for the evolving partially colored graph.] However, this algorithm is inherently sequential in nature: we first need to fix a partial coloring and then consider the next edge. Using limited independence we can derandomize the algorithm in a way that admits efficient parallelization, as we shall now see.

Set $r = \binom{n}{2}$ and $d = \binom{k}{2}$.
Suppose we select the edge colors using not mutually independent coin flips, but instead using a family of r d-

wise independent {red, blue}-valued random variables. Then the calculation of the expectation $\mathrm{E}[X]$ remains unchanged, since it only requires that groups of $d = \binom{k}{2}$ edges (corresponding to cliques) are independent.

How can we construct such a family? We use a generalization of the construction in Claim 10.4, as follows. Let $q > r$ be prime and pick d values $a_0, a_1, \cdots, a_{d-1} \in \mathbb{Z}_q$ u.a.r. It is left as an **exercise** to show that the family

$$\left\{ \sum_{i=0}^{d-1} a_i x^i : x \in \mathbb{Z}_q \right\}$$

consists of q d-wise independent, uniform random variables. We can use any reasonable method to project these down uniformly onto $\{\text{red}, \text{blue}\}$.

Now notice that the total number of points in our sample space is only $q^d \simeq \binom{n}{2}^{\binom{k}{2}} = n^{O(k^2)}$. Therefore, rather than generating samples at random, we can exhaustively try all the possible sample points in polynomial time, thereby yielding an efficient *deterministic* algorithm (for any constant k). Moreover, this algorithm can easily be parallelized as follows:

set $r = \binom{n}{2}$ and $d = \binom{k}{2}$
for each of the $n^{O(k^2)}$ sample points in parallel do
    use the sample point to color the edges of the graph
    for each of the $O(n^k)$ k-cliques in the graph in parallel do
        test if the clique is monochromatic
    add outputs to find the total number of monochromatic k-cliques in the graph
output any coloring with at most $f(k)$ monochromatic k-cliques

The above algorithm requires $n^{O(k^2)}$ processors. Note that we also have to parallelize the generation of the edge colors and the addition at the end. The edge colors are obtained by evaluating a degree-d polynomial at r points over $\mathbf{Z}_q$. Each evaluation takes $O(\log n)$ time (d is a constant), and since we already have more than r processors per sample point we can perform the evaluations in parallel. Finally, the addition of $O(n^k)$ 0-1 values can be performed in $O(\log n)$ time in parallel.

Thus for any fixed k, the entire algorithm runs on $n^{O(k^2)}$ processors in time $O(\log n)$.

10.2 Universal hashing

Suppose we have a universe U of keys and a subset $S \subset U$ with $|S| \ll |U|$. (Think, for example, of U being the set of all 2^{32} 32-bit strings, and S some subset of them that is being used by some algorithm.) We want to store the items of S in a data structure of size roughly $|S|$ in such a way that we can efficiently perform the operations Add, Delete and Search. (Such a data structure is called a "Dictionary.")

A standard solution to this problem is accomplished by using a *hash function* $h : U \to T$, where T is a hash table of size roughly $|S|$.

For a given hash function h, a *collision* is a pair of elements $x, y \in S$, with $x \neq y$, for which $h(x) = h(y)$. We want to minimize the number of collisions, because when elements land in the same location we have to store them in (say) a linked list, and the search time at that location will be proportional to the length of this list.

Obviously, for any fixed hash function h we can always choose a bad set $S \subset U$ such that the number of collisions is large. (Exercise: Why?) In order to get around this problem, we could choose our hash

function h *randomly*, so that the location of each stored word is independent and identically distributed. In practice, however, completely random hash functions are too expensive to compute and store: if $|U| = M$ and $|T| = n$, there are n^M functions from U to T, so each function requires at least $O(M \log n)$ bits just to write it down.

Fortunately, it turns out that choosing our hash function from a *pairwise* independent family is enough in this application. In the hashing context, pairwise independent functions are usually referred to as "2-universal hash functions" [CW79]:

Definition 10.5 *A family $\mathfrak{H}$ of functions from U to T is* 2-universal *if $\forall x, y \in U$ with $x \neq y$ we have*

$$\Pr_{h \in \mathfrak{H}}[h(x) = h(y)] \leq 1/n,$$

where $n = |T|$.

Note the slight difference from the definition of pairwise independence: the latter would require that the collision probability be *exactly* $1/n$.

Example : The family $\{f_{a,b}(x) : x \in \mathbb{Z}_q\}$ from Claim 10.4 is 2-universal. (This family maps $\mathbb{Z}_q$ to $\mathbb{Z}_q$; in the hashing application, we would take q to be the size of U and then take the output modulo n, where $n = |T|$. Exercise: Check that the result is 2-universal.)

Note that we can construct and represent 2-universal hash functions using only $O(\log M)$ bits, which is the same number of bits as in a single key.

The main idea behind the use of 2-universal hash functions lies in the fact that for any x we have
E[number of collisions with x] $\leq (|S| - 1) \times \frac{1}{n} < 1$,
assuming that $n \geq |S|$. Thus the *expected* search time for any element is constant. However, this does not give us a bound on the worst-case search time.

This motivates the problem of designing a data structure with worst-case constant search time. The following solution, due to Fredman, Komlós and Szemerédi [FKS84], applies only to the case of a *static* dictionary. In a static dictionary, items are inserted once at the beginning, and there are no subsequent deletions or insertions. Only lookups are performed subsequently.

Theorem 10.6 *For a static dictionary, we can achieve no collisions (and hence worst-case constant search time) using a hash table of size $O(|S|)$.*

Proof: The approach is known as "double-hashing", and uses a two-level scheme as follows. First, we hash the set S into a hash table with n bins using a hash function from a 2-universal family. Let b_i be the number of elements that end up in bin i. Some of these bins will have collisions. For each such bin i, we pick a second hash function h_i from another 2-universal family and create a secondary hash table for this bin. We ensure that there are no collisions in these final tables.

We consider first the secondary hash functions, which rely on the following:

Claim 10.7 *If we use a 2-universal hash family to map b items to a table of size b^2, then* $\Pr[\exists$ a collision$] \leq \frac{1}{2}$.

To prove this claim, simply note that

$$\text{E[number of collisions]} \leq \binom{b}{2} \times \frac{1}{b^2} \leq \frac{1}{2}$$

and then apply Markov's inequality.

Observe that, in order to find a hash function with no collisions, we may simply try hash functions chosen uniformly at random from the family of 2-universal hash functions until we find one with no collisions. On average, we will need to try at most two hash functions. Thus we will choose the secondary hash table for bin i to have size b_i^2.

We now turn to the question of choosing the primary hash function h. Note that this function must ensure that $\sum b_i^2 = O(|S|)$, since this is the total size of all the secondary tables.

Claim 10.8 *If we use a 2-universal hash family to map $|S|$ items to a table of size $n \geq |S|$ then* $\Pr[\sum b_i^2 \geq 4|S|] \leq 1/2$.

To prove this claim, note that the total number of collisions is $\sum_i \binom{b_i}{2} = \frac{1}{2}(\sum_i b_i^2 - \sum_i b_i) = \frac{1}{2}(\sum_i b_i^2 - |S|)$. Hence, taking expectations:

$$\mathrm{E}\big[\sum b_i^2\big] = 2 \times \mathrm{E}[\text{number of collisions}] + |S| \leq 2\binom{|S|}{2} \times 1/n + |S| \leq 2|S|,$$

where the first inequality follows from the 2-universal property. The claim then follows by Markov's inequality.

As before, we can simply try random hash functions h from the 2-universal family until we find one with the desired property. We expect to try at most two of them.

In the final data structure, the cells of the primary hash table store the descriptions of the secondary hash functions h_i. Note that the total number of bits needed for these is just $O(\sum b_i \log b_i)$, which is again $O(|S|)$. The whole structure therefore has size $O(|S|)$, as desired. ■

References

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. "Deterministic simulation in logspace," *Proceedings of the 19th ACM Symposium on Theory of Computing*, 1987, pp. 132–140.

[CW79] J.L. Carter and M.N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences* **18** (1979), pp. 143–154.

[CG89] B. Chor and O. Goldreich, "On the power of two-point sampling," *Journal of Complexity* **5** (1989), pp. 96–106.

[FKS84] M.L. Fredman, J. Komlós and E. Szemerédi, "Storing a sparse table with $O(1)$ worst case access time," *Journal of the ACM* **31** (1984), pp. 538–544.

[J74] A. Joffe, "On a set of almost deterministic k-independent random variables," *Annals of Probability* **2** (1974), pp. 161–162.

[Z96] D. Zuckerman, "Simulating BPP using a general weak random source," *Algorithmica* **16** (1996), pp. 367–391.