6.842 Randomness and Computation 2012-02-22

Lecture 5

Lecturer: Ronitt Rubinfeld *Scribe: Michael Forbes*

1 Overview

Today we will define the notion of a pairwise independent hash function, and discuss its applications in randomness-efficient error reduction and interactive proofs. Randomness-efficient error reduction is the topic of studying algorithms for amplifying the success probability (equivalently, reducing the error probability) of an RP algorithm by running the algorithm many times, each with a new sample of randomness. However, as the goal is to minimize the use of randomness, we will not use truly random samples, but rather pseudorandom samples. In particular, we will show that pairwise independent samples already give a non-trivial error-reduction procedure.

The second application is to interactive proofs, which are interactive communication protocols for one player (the prover) to "prove" a statement to another player (the verifier). These players are allowed randomness when executing the protocol, and there are two models of this: the "public" model where all random coin tosses are revealed publicly, and the "private" model, where random coin tosses can remain private (but still may be revealed if the communication protocol requires it). Intuitively, it seems clear that any protocol in the private model can simulate a protocol in the public model. More surprising is that the public model can also simulate the private model, and the proof of this will use pairwise independent hash functions. Today we will introduce the model, and next time we will discuss this application of pairwise independence.

2 Pairwise Independent Hash Functions

2.1 Extending the construction from last time

Recall the definition from last time of pairwise independent random variables.

Definition 1 *Let S be a finite set, and let $X_1, \ldots, X_n$ be random variables assuming values in S. Then $X_1, \ldots, X_k$ are **pairwise independent** if for all $i \neq j$ and $a, b \in S$,*

$$\Pr[X_i = a \wedge X_j = b] = 1/|S|^2$$

Here, as is the convention, we define pairwise independence so that the random variables are also (individually) uniform over the sample space. That is, for all i and a, $\Pr[X_i = a] = 1/|S|$, which can be seen by summing over all possible b for some fixed $j \neq i$. Clearly in the above definition we need to restrict $i \neq j$ for this definition to make sense.

Last time, we also showed a simple construction of pairwise independent bits.

Claim 2 *Let $y_1, \ldots, y_n$ be uniform random bits. For each non-empty subset $S \subseteq [n]$, define $X_S = \oplus_{i \in S} y_i$. Then the bits $\{X_S\}_{\emptyset \neq S \subseteq [n]}$ are pairwise independent.*

This claim shows how to stretch n truly random bits to $2^n - 1$ pairwise independent bits. Recall that this exponential stretch is important, as in derandomization applications we will enumerate over the entire sample space, which is exponential in the number of random bits needed to sample from that space. So using n bits in our construction will require 2^n in enumeration. In our application to MAX-CUT, we could take n so that $2^n - 1 \geq m$, where m was the number of vertices in the graph, so this exponential enumeration is efficient as we are only really using $\lg m$ random bits.

In the MAX-CUT application it was sufficient to get pairwise independent *bits*. But what if we wanted to have pairwise independent random variables that assume more than just 2 values? We now outline an easy way to get this, using our above construction.

Claim 3 *Let $\{X_{i,1}\}_{i=1}^n, \ldots, \{X_{i,k}\}_{i=1}^n$ be independent collections of pairwise independent random variables assuming values in the set S. That is, each collection of $\{X_{i,j}\}_{i=1}^n$ is pairwise independent, and a variable $X_{i,j}$ is independent of the all variables $\{X_{i',j'}\}_{i\in[n],j\neq j'}$. Then the variables $\{(X_{i,1},\ldots,X_{i,k})\}_{i=1}^n$ are pairwise independent, assuming values in the set S^k.*

Proof This is via direct calculation.

$$\begin{aligned}
&\Pr[(X_{i,1},\ldots,X_{i,k}) = (a_1,\ldots,a_k) \wedge (X_{i',1},\ldots,X_{i',k}) = (b_1,\ldots,b_k)] \\
&= \Pr[X_{i,1} = a_1 \wedge \cdots \wedge X_{i,k} = a_k \wedge X_{i',1} = b_1 \wedge \cdots \wedge X_{i',k} = b_k] \\
&= \Pr[X_{i,1} = a_1 \wedge X_{i',1} = b_1 \wedge \cdots \wedge X_{i,k} = a_k \wedge X_{i',k} = b_k]
\end{aligned}$$

By independence of the collections,

$$= \Pr[X_{i,1} = a_1 \wedge X_{i',1} = b_1] \cdots \Pr[X_{i,k} = a_k \wedge X_{i',k} = b_k]$$

By pairwise independence,

$$= (1/|S|^2)^k = 1/|S^k|^2$$

■

Thus, as we have constructed pairwise independent variables over bits, the above construction gives pairwise independent variables over $\{0,1\}^k$ for any k. Using the binary expansion, this gives such random variables over the set $[2^k]$. Let's now discuss the randomness efficiency of this construction. In particular, let's suppose we want to construct n pairwise random variables, each over the set $[n]$. The above construction would then take $\approx \lg n$ independent copies of the construction for bits, which itself takes $\approx \lg n$ random bits. So this gives $\approx \lg^2 n$ random bits used. While this is much smaller than the $\approx n \lg n$ random bits needed for independent random variables, this is too large for derandomization-via-enumeration, as $2^{\lg^2 n} = n^{\lg n}$, which is super-polynomial. Thus, better constructions are needed.

2.2 Hash Functions

Before we discuss more constructions, we will discuss a stronger notion than pairwise independence: that of a pairwise independent hash function. When thinking of pairwise independent random variables (over bits), we had the following picture in mind: let M be the $2^n \times n$ matrix consisting of all n-bit bit-strings. A set of n-wise random variables is really just a way to sample one row uniformly from this matrix. A set of pairwise independent random variables will also sample rows from this matrix, but will sample from a much smaller set of rows (as eventually we will want to enumerate over every row, so we want the number of such rows to be small). The property that we wish to ensure is that for every two columns, sampling from this smaller set of rows yields a uniform distribution over all 4 possible 2-bit strings. So in summary we get a subset of rows of A that approximates the entire space of rows of A, at least when looking at pairs of columns.

While this viewpoint is accurate, it obscures one major point: that of efficiency/explicitness. That is, we can describe a set of pairwise independent random variables by the set of rows of the matrix A that they sample from. However, what if we want to just sample from only one of the random variables? To have to construct the entire submatrix of A necessarily takes a lot of time

(as this submatrix is large). So it would seem that instead of sampling each row of this submatrix, we really want a way to individually sample the random variables. This naturally leads us to the viewpoint of a hash function. We can think of a row of the matrix A as a function from the index i to the value of the random variable X_i. When we sample a row from A, we are really just sampling a random function. The importance of this notion is that now we can discuss the efficiency of this function, which is clearly important in algorithmic discussions. We now formalize this notion in the following definition.

Definition 4 *A family of functions $\mathcal{H} = \{h | h : [N] \to [M]\}$ is called a **family of pairwise independent hash functions** if for all $i \neq j \in N$ and $k, \ell \in M$,*

$$\Pr_{h \leftarrow \mathcal{H}}[h(i) = k \wedge h(j) = \ell] = 1/M^2$$

It should be clear that the random variables $h(i)$ and $h(j)$ are pairwise independent. Note that the input and output are sized $\lg N$ and $\lg M$ respectively, so the functions h should be computable in some polynomial of these bounds to be considered efficient.

We remark here that our notion of a pairwise independent hash family is also known as a "strongly universal hash family". There is also a "weak" version, which simply guarantees that for $i \neq j$, $\Pr[h(i) = h(j)] \leq 1/M$ (note that in the above, this is met with equality). Often, this weaker notion is sufficient in practice. For example, one of the original motivations for hash functions was for *hashing* a small number (say $\sqrt{M}$) elements from a large universe (eg. $[N]$) into a small table of size $[M]$. One hopes to support insert and lookup queries in this table. If two elements are inserted into the same place in the table, then the lookup on these elements is longer as the lookup must query all elements mapped into one place in the table. One solution would be to use a completely random function when mapping into the table. However, this makes it very difficult to use, as such a random function takes lots of space just to write down (and it must be written down, so that the lookup query can probe the same place as the insert query). So we can then attempt to use a function which is "sufficiently random". A natural candidate is a pairwise independent hash family, for we are simply seeking to minimize collisions, and collisions are pairwise events, so the statistics will be the same. As a consequence, pairwise independent hash families

2.3 Constructing Hash Functions

We now construct a family of hash functions. Recall that for any prime p, the set $\mathbb{Z}_p = \{0, \ldots, p-1\}$ forms a field, which means that addition and multiplication work just as they do over the integers, and each non-zero element has a multiplicative inverse. (Contrast this to $\mathbb{Z}_n$, for n composite, where addition and multiplication still work, but elements may not have a multiplicative inverse, for example $2 \cdot 3 = 0$ over $\mathbb{Z}_6$.) We now consider the following construction of pairwise independent hash functions.

Claim 5 *Let p be a prime. For $a, b \in \mathbb{Z}_p$, define $h_{a,b} : \mathbb{Z}_p \to \mathbb{Z}_p$ by $h_{a,b}(x) = ax + b$. Then the collection of functions $\mathcal{H} = \{h_{a,b} | a, b \in \mathbb{Z}_p\}$ is a pairwise independent hash family.*

Proof Let $x \neq y \in \mathbb{Z}_p$ and $c, d \in \mathbb{Z}_p$. Then

$$\Pr_{a,b}[h_{a,b}(x) = c \wedge h_{a,b}(y) = d] = \Pr\left[\begin{bmatrix} x & 1 \\ y & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}\right]$$

Using the matrix inverse,

$$\begin{aligned}
&= \Pr\left[\left(\frac{1}{x-y}\cdot\begin{bmatrix}1 & -1\\ -y & x\end{bmatrix}\right)\begin{bmatrix}x & 1\\ y & 1\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix} = \left(\frac{1}{x-y}\cdot\begin{bmatrix}1 & -1\\ -y & x\end{bmatrix}\right)\begin{bmatrix}c\\ d\end{bmatrix}\right]\\
&= \Pr\left[\begin{bmatrix}a\\ b\end{bmatrix} = \left(\frac{1}{x-y}\cdot\begin{bmatrix}1 & -1\\ -y & x\end{bmatrix}\right)\begin{bmatrix}c\\ d\end{bmatrix}\right]\\
&= 1/p^2
\end{aligned}$$

as desired. ■

As a quick example, suppose $p = 3$. Then we get the following table, where each row is a function from $\mathbb{Z}_3 \to \mathbb{Z}_3$.

$$\begin{bmatrix}
0 & 0 & 0\\
1 & 1 & 1\\
2 & 2 & 2\\
0 & 1 & 2\\
1 & 2 & 0\\
2 & 0 & 1\\
0 & 2 & 1\\
1 & 0 & 2\\
2 & 1 & 0
\end{bmatrix}$$

We now make some remarks. The first is that the function $h_{a,b}$ is always efficiently computable, as we just need several operations modulo p. The second is that "$a = 0$" is a possibility. What is slightly confusing about that possibility is that $h_{0,b}(x) = b$ for all x. Intuitively this is undesirable, as constant functions are clearly undesirable from a practical perspective of being random (and especially bad when considering the application to hashing). However, if we disallowed "$a = 0$" then the family would not be pairwise independent. That is, for $x \neq y$, $h_{a,b}(x) = h_{a,b}(y)$ iff $a = 0$. This can be seen in the above example, where the first three rows correspond to $a = 0$. The third comment is that this construction readily generalizes to k-wise independent hash families, where we take $h_{a_{k-1},\ldots,a_0}(x) = \sum_{i=0}^{k-1} a_i x^i$. The proof that this construction is k-wise independent follows from the fact that the Vandermonde matrix

$$\begin{bmatrix}
1 & 1 & \ldots & 1\\
x_1 & x_2 & \ldots & x_k\\
x_1^2 & x_2^2 & \ldots & x_k^2\\
\vdots & \vdots & \ddots & \vdots\\
x_1^{k-1} & x_2^{k-1} & \ldots & x_k^{k-1}
\end{bmatrix}$$

is invertible for distinct $x_1, \ldots, x_k$, which can also be seen as the statement that polynomial interpolation of degree $< k$ polynomials is always uniquely possible from k evaluation points.

Additionally, we observe that this construction takes $2 \lg p$ random bits, and its extension to k-wise independence takes $k \lg p$ random bits. For small k, this is certainly within our budget to derandomize via enumeration (as opposed to the other construction given above, which we cannot derandomize efficiently).

The above construction works when the size of the set we want to sample from is prime. One can also apply this construction over any finite field. In particular, if we do so to a field of size q, where q is a power of two, then we get the following fact (after some truncation). For a reference, see `http://people.seas.harvard.edu/~salil/pseudorandomness/`

Fact 6 *There is an explicit family $\mathcal{H}$ of pairwise independent hash functions from $\{0,1\}^n \to \{0,1\}^m$ constructed using $\max n, m + n$ random bits, computable in $\mathrm{poly}(n, m)$ time.*

One can also try to truncate the results modulo p to lie in the set $\{0,1\}^n$, but this ruins uniformity and also requires finding a large prime in polynomial time, which we only know how to do using randomness.

2.4 Randomness-efficient error reduction, from pairwise independence

We now give an application of pairwise independent hash functions. To do so, we first need to show a tail inequality for pairwise independent random variables.

Claim 7 (Pairwise Independent Tail Inequality) *Let $X_1, \ldots, X_t$ be pairwise random variables with values in $[0,1]$. Define $X := \sum_i X_i$. Then $\Pr[|X - \mathbb{E}[X]| \geq \epsilon t] \leq \frac{1}{\epsilon^2 t}$.*

Proof Note that $\mathbb{E}[X] = \sum_i \mathbb{E}[X_i]$ and

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathbb{E}[(\sum_i X_i)^2] - \mathbb{E}[\sum_i X_i]^2 = \sum_i (\mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2) + \sum_{i \neq j} (\mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j])$$

Invoking pairwise independence,

$$= \sum_i \mathrm{Var}(X_i) + \sum_{i \neq j} (\mathbb{E}[X_i]\mathbb{E}[X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]) = \sum_i \mathrm{Var}(X_i)$$

So as $\mathrm{Var}(X_i) = \mathbb{E}[(X_i - \mathbb{E}[X_i])^2]$, and X_i assumes values in $[0,1]$, it follows that $\mathrm{Var}(X_i) \leq 1$ and so $\mathrm{Var}(X) \leq t$. Thus, applying the Chebyschev inequality we get that $\Pr[|X - \mathbb{E}[X]| \geq \epsilon t] \leq \frac{1}{\epsilon^2 t}$ as desired. ■

We remark here that if the X_i are Bernoulli random variables, then $\mathrm{Var}(X_i) \leq 1/4$, as the variance is maximized when $p = 1/2$. This can be used to slightly improve the above bound.

What this bound is saying is that although the X_i's are not truly independent, their sum does concentrate around its expectation. This will be useful for us, as it says that if we want to estimate the average of some function f, then if we take enough pairwise independent samples $f(X_i)$, then the average of these samples will concentrate around the true average. The key point here is that we can save on the number of random bits used because we only needed to generate pairwise independent samples instead of truly independent samples.

Let's now apply this to amplification of RP algorithms. Let's first review amplification using truly independent random bits.

Claim 8 *Let A be an RP algorithm for a language L that uses at most $r(n)$ random bits and runs in time $t(n)$, so that*

- $x \in L \implies \Pr_r[A(x,r) = 1] \geq 1/2$
- $x \notin L \implies \Pr_r[A(x,r) = 1] = 0$

Then there is an RP algorithm A' for L that uses at most $k \cdot r(n)$ random bits, runs in time $k \cdot t(n)$, so that

- $x \in L \implies \Pr_r[A(x,r) = 1] \geq 1 - 1/2^k$
- $x \notin L \implies \Pr_r[A(x,r) = 1] = 0$

Proof The algorithm A' will simply run $A(x, r_1), \ldots, A(x, r_k)$, for $r_1, \ldots, r_k \in \{0,1\}^{r(n)}$ uniformly at random. A' will output 1 iff $A(x, r_i)$ for some i. The runtime and random bit usage are straightforward to analyze. Now observe that if $x \notin L$ then $A(x, r_i) = 0$ for all i. If $x \in L$, then $\Pr[A(x, r_i) = 1] \geq 1/2$ for each i, so by independence, the probability that at least one i is so that $A(x, r_i) = 1$ is at least $1 - 1/2^k$. ■

We now study how we can improve the above amplification, by choosing the r_i to be pairwise independent, instead of fully independent.

Claim 9 *Let A be an RP algorithm for a language L that uses at most $r(n)$ random bits and runs in time $t(n)$, so that*

- $x \in L \implies \Pr_r[A(x, r) = 1] \geq 1/2$
- $x \notin L \implies \Pr_r[A(x, r) = 1] = 0$

Then there is an RP algorithm A' for L that uses at most $O(k + r(n))$ random bits, runs in time $O(2^k \cdot t(n))$, so that

- $x \in L \implies \Pr_r[A(x, r) = 1] \geq 1 - 1/2^{k-2}$
- $x \notin L \implies \Pr_r[A(x, r) = 1] = 0$

Proof The algorithm A' will first sample from a pairwise independent hash family $\mathcal{H} : \{0,1\}^k \to \{0,1\}^{r(n)}$, and by the above fact, this will take $O(k + r(n))$ random bits. This yields the 2^k strings $r_1, \ldots, r_{2^k}$, which are pairwise independent. A' will then output 1 iff $A(x, r_i) = 1$ for some i. From here, the bounds on time and randomness used are clear. We now analyze the acceptance probabilities. As usual, if $x \notin L$, then A' will never accept. If $x \in L$, then we can define X_i to be the indicator event for $A(x, r_i) = 1$, so that $\mathbb{E}[X_i] \geq 1/2$. Thus, A' accepts x iff $X := \sum_i X_i$ is > 0. Seeing that $\mathbb{E}[X] \geq 1/2$, we now observe that $\Pr[X = 0] = \Pr[|X - \mathbb{E}[X]| \geq \frac{1}{2} \cdot 2^k] \leq \frac{4}{2^k}$, using the pairwise independent tail inequality. This yields the result. ■

So this result shows that we can dramatically save on the number of random bits if we want an error probability of $1/2^k$, reducing from the original $k \cdot r(n)$ to $O(k + r(n))$. However, this comes with the price in runtime. Later in the class, we will show (using a technique called an expander walk) how to get the best of both of these techniques. That is, we will see how to transform the RP algorithm A into one that uses $r(n) + O(k)$ random bits, in time $O(kt(n))$, and achieves an error probability of $\approx 1/2^k$.

3 Interactive Proofs

A second application of pairwise independent hash functions is to the theory of interactive proofs. Interactive proofs can be motivated by the following example. Suppose an individual, named Merlin, claims to have the extraordinary ability to differentiate the soft-drinks Coke and Pepsi. Arthur, a mere mortal, does not have this ability and is skeptical of Merlin's powers. How can Merlin prove to Arthur that he has this great power? Here is such a protocol. Arthur can prepare two samples of drink, one of Coke, and one of Pepsi. He then tosses a coin and permutes these drinks, and then hands them to Merlin in the permuted order. Merlin must then identify which drink is Coke, and which is Pepsi. It should be clear from the setup that if Merlin cannot differentiate Coke and Pepsi, then his chance of identifying the correct drink is $1/2$ and is otherwise the chance is 1 (in the real world, one might have intermediate probabilities, as one might have an intermediate skill at this prediction, but let us ignore this). So after t trials of this protocol, the chance that Merlin has successfully predicted each drink is at most $1/2^t$ if he does not have prediction powers, and is 1

if he does have prediction powers. If we let t be sufficiently large, then if Merlin predicts all of the drinks correctly this can be considered to be "proof" that he has such a power.

The above example is a so-called "private-coin" model of interaction, as it was very important that Merlin not see the coins that Arthur tosses, as those coins determine which drink is Coke and which is Pepsi. That is, if Arthur had no way to secretly permute the drinks, the above protocol would fail. On the face of it, it seems very difficult in the "public coin" model to prove that Coke and Pepsi can be differentiated. However, next time we will see how this is possible, using pairwise independent hash functions.