COSC 544 Probabilistic Proof Systems 9/5/17

The Power of Randomness: Fingerprinting and Freivalds' Algorithm

Lecturer: Justin Thaler

1 Fingerprinting

1.1 The Setting

Alice and Bob live across the country from each other. They each hold a very large file, each consisting of n characters (for concreteness, suppose that these are ASCII characters, so there are $m = 128$ possible characters). Let us denote Alice's file as the sequence of characters $(a_1, \ldots, a_n)$, and Bob's as $(b_1, \ldots, b_n)$. Their goal is to determine whether their files are *equal*, i.e., whether $a_i = b_i$ for all $i = 1, \ldots, n$. Since the files are large, they would like to minimize the *communication*, i.e., Alice would like to send as little information about her file to Bob as possible.

A trivial solution to this problem is for Alice to send her entire file to Bob, and Bob can check whether $a_i = b_i$ for all $i = 1, \ldots, n$. But this requires Alice to send all n characters to Bob, which is prohibitive if n is very large. It turns out that no *deterministic* procedure can send less information than this trivial solution.

However, we will see that if Alice and Bob are allowed to execute a *randomized* procedure (which might output the wrong answer with some tiny probability, say at most 0.0001), then they can get away with a much smaller amount of communication.

1.2 Reed-Solomon Fingerprinting

1.2.1 The Communication Protocol

The High-Level Idea. The rough idea is that Alice is going to pick a hash function h at random from a (small) family of hash functions $\mathcal{H}$. We will think of $h(x)$ as a very short "fingerprint" of x. By fingerprint, we mean that $h(x)$ is a "nearly unique identifier" for x, in the sense that for any $y \neq x$, the fingerprints of x and y differ with high probability over the random choice of h, i.e.,

$$\text{for all} x \neq y, \Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq 0.0001.$$

Rather than sending a to Bob in full, Alice sends h and $h(a)$ to Bob. Bob checks whether $h(a) = h(b)$. If $h(a) \neq h(b)$, then Bob *knows* that $a \neq b$, while if $h(a) = h(b)$, then Bob can be very confident that $a = b$.

The Details. To make the above outline concrete, fix a prime number $p > \max\{m, n^2\}$, and let $\mathbb{F}_p$ denote the set of integers modulo p. For the remainder of this section, we assume that all arithmetic is done *modulo* p without further mention. So, for example, if $p = 17$, then $2 \cdot 3^2 + 4 = 22 (\bmod 17) = 5$.

For each $r \in \mathbb{F}_p$, define $h_r(a_1, \ldots, a_n) = \sum_{i=1}^{n} a_i \cdot r^i$. The family $\mathcal{H}$ of hash functions we will consider is

$$\mathcal{H} = \{h_r : r \in \mathbb{F}_p\}.$$

Intuitively, each hash function h_r interprets its input $(a_1, \ldots, a_n)$ as the coefficients of a degree n polynomial, and outputs the polynomial evaluated at r.

That is, Alice picks a random element r from $\mathbb{F}_p$, computes $v = h_r(a)$, and sends v and r to Bob. Bob outputs EQUAL if $v = h_r(b)$, and outputs NOT-EQUAL otherwise.

1.2.2 The Analysis

We now prove that this protocol outputs the correct answer with very high probability. In particular:

- If $a_i = b_i$ for all $i = 1, \dots, n$, then Bob outputs EQUAL for every possible choice of r.
- If there is even one i such that $a_i \neq b_i$, then Bob outputs NOT-EQUAL with probability at least $1 - n/p$, which is at least $1 - 1/n$ by choice of $p > n^2$.

The first property is easy to see: if $a = b$, then obviously $h_r(a) = h_r(b)$ for every possible choice of r. The second property relies on the following crucial fact, whose validity we justify later in Section 1.2.5.

Fact 1. *For any two distinct (i.e., unequal) polynomials p_a, p_b of degree at most n with coefficients in $\mathbb{F}_p$, $p_a(x) = p_b(x)$ for at most n values of x in $\mathbb{F}_p$.*

Let $p_a(x) = \sum_i a_i \cdot x^i$ and similarly $p_b(x) = \sum_i b_i \cdot x^i$. Observe that both p_a and p_b are polynomials in x of degree at most n. And the value v that Alice sends to Bob in the communication protocol is precisely $p_a(r)$, and Bob compares this value to $p_b(r)$.

By Fact 1, there are at most n values of r such that $p_a(r) = p_b(r)$. Since r is chosen at random from $\mathbb{F}_p$, the probability that Alice picks such an r is thus at most n/p. Hence, Bob outputs NOT-EQUAL with probability at least $1 - n/p$ (where the probability is over the random choice of r).

1.2.3 Cost of the Protocol

Alice sends only two elements of $\mathbb{F}_p$, namely v and r, to Bob in the above protocol. In terms of bits, this is $O(\log n)$ bits assuming $p \leq n^c$ for a constant c. This is an *exponential improvement* over the $n \cdot \log m$ bits sent in the deterministic protocol. This is an impressive demonstration of the power of randomness.

1.2.4 Discussion

We refer to the above protocol Reed-Solomon fingerprinting because $p_a(r)$ is actually a random entry in an *error-corrected encoding* of the vector $(a_1, \dots, a_n)$ called the Reed-Solomon encoding. Several other fingerprinting methods are known. Indeed, all that we really require of the hash family $\mathcal{H}$ used in the protocol above is that for any $x \neq y$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)]$ is small. Many hash families are known to satisfy this property, but Reed-Solomon fingerprinting will prove particularly relevant in our study of probabilistic proof systems owing to its algebraic structure.

A few sentences on finite fields. For prime p, $\mathbb{F}_p$ is an example of a *field*, which is any set equipped with addition, subtraction, multiplication, and division operations. So, for example, the set of real numbers is a field, because for any two reals numbers c and d, $c+d$, $c-d$, $c \cdot d$, and (assuming $d \neq 0$) c/d are themselves all real numbers. The same holds for the set of complex numbers, and the set of rational numbers. In contrast, the set of integers is *not* a field, since dividing two integers does not necessarily yield another integer.

$\mathbb{F}_p$ is also a field (a *finite* one). Here, the field operations are simply addition, subtraction, multiplication, and division modulo p. What we mean by division modulo p requires some explanation: for every $a \in \mathbb{F}_p$, there is a unique element $a^{-1} \in \mathbb{F}_p$ such that $a \cdot a^{-1} = 1$. For example, if $p = 5$ and $a = 3$, then $a^{-1} = 2$, since $3 \cdot 2 \pmod 5 = 6 \pmod 5 = 1$. Division by a in $\mathbb{F}_p$ refers to multiplication by a^{-1}. So if $p = 5$, then in $\mathbb{F}_p$, $4/3 = 4 \cdot 3^{-1} = 4 \cdot 2 = 3$.

1.2.5 Establishing Fact 1

Fact 1 is implied by (in fact, equivalent to) the following fact.

Fact 2. *Any non-zero polynomial of degree at most n over any field has at most n roots.*

A simple proof of Fact 2 can be found online at [hp]. To see that Fact 2 implies Fact 1, observe that if $p_a \neq p_b$ are polynomials of degree at most n, and $p_a(x) = p_b(x)$ for more than n values of $x \in \mathbf{F}_p$, then $p_a - p_b$ is a nonzero polynomial of degree at most n with with more than n roots.

2 Freivalds' Algorithm

2.1 The Setting

Suppose we are given as input two $n \times n$ matrices A and B over $\mathbb{F}_p$, where $p > n^2$ is a prime number. Our goal is to compute the product matrix $A \cdot B$.

Asymptotically, the fastest known algorithm for accomplishing this task is very complicated, and runs in time $O(n^{2.3728639})$ time [LG14]. Moreover, the algorithm is not practical.

But for the purposes of a course on probabilistic proof systems, the relevant question is not how fast can we multiply two matrices—it's how efficiently can one *verify* that two matrices were multiplied correctly. In particular, can verifying the output of a matrix multiplication problem be done faster than the fastest known algorithm for actually multiplying the matrices? The answer, given by Freivalds in 1977 [Fre77], is yes.

Formally, suppose someone hands us a matrix C, and we want to check whether or not $C = A \cdot B$. Here is a very sample randomized algorithm that will let us perform this check in $O(n^2)$ time (this is only a constant factor more time than what is required to simply read the matrices A, B, and C).

2.2 The Algorithm

First, choose a random $r \in \mathbb{F}_p$, and let $x = (r, r^2, \ldots, r^n)$. Then compute $y = Cx$ and $z = A \cdot Bx$, outputting YES if $y = z$ and NO otherwise.

2.3 Runtime

We claim that the entire algorithm runs in time $O(n^2)$. It is easy to see that generating the vector $x = (r, r^2, \ldots, r^n)$ can be done with $O(n)$ total multiplications (r^2 can be computed as $r \cdot r$, then r^3 can be computed as $r \cdot r^2$, then r^4 as $r \cdot r^3$, and so on). Since multiplying an $n \times n$ matrix by an n-dimensional vector can be done in $O(n^2)$ time, the remainder of the algorithm runs in $O(n^2)$ time: computing y involves multiplying C by the vector x, and computing $A \cdot Bx$ involves multiplying B by y to get a vector $w = By$, and then multiplying A by w to compute $A \cdot Bx$.

2.4 Analysis

Let $D = A \cdot B$. We claim that the above algorithm satisfies the following two conditions:

- If $C = D$, then the algorithms outputs YES for every possible choice of r.
- If there is even one $(i, j) \in [n] \times [n]$ such that $C_{i,j} \neq D_{i,j}$, then Bob outputs NO with probability at least $1 - n/p$.

The first property is easy to see: if $C = D$, then clearly $Cx = Dx$ for all vectors x.

The see the second property, suppose that $C \neq D$, and let C_i and D_i denote the ith row of C and D respectively. Obviously, since $C \neq D$, there is some row i such that $C_i \neq D_i$. Observe that $(Cx)_i$ is precisely $p_{C_i}(r)$, the Reed-Solomon fingerprint of C_i as in the previous section. Similarly, $(A \cdot B \cdot x)_i = p_{D_i}(r)$. Hence, by the analysis of the previous lecture, the probability that $(Cx)_i \neq (A \cdot B \cdot x)_i \geq 1 - n/p$, and in this event the algorithm outputs NO.

2.5 Discussion

Whereas fingerprinting saved communication compared to a deterministic protocol, Freivalds' algorithm saves *runtime* compared to the best known deterministic algorithm. We can think of Freivalds' algorithm as our first probabilistic proof system: here, the proof is simply the answer C itself, and the $O(n^2)$-time verification procedure simply checks whether $Cx = A \cdot Bx$.

Freivalds actually described his algorithm with a perfectly random vector $x \in \mathbb{F}_p^n$, rather than $x = (r, r^2, \ldots, r^n)$ for a random $r \in \mathbb{F}_p$. We chose $x = (r, r^2, \ldots, r^n)$ to ensure that $(Cx)_i$ is a Reed-Solomon fingerprint of row i of C, thereby allowing us to invoke the analysis from Section 1.

References

[Fre77] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, volume 839, page 842, 1977. [p. 2: The Power of Randomness: Fingerprinting and Freivalds' Algorithm-3]

[hp] Dan Petersen (https://math.stackexchange.com/users/677/dan petersen). How to prove that a polynomial of degree n has at most n roots? Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/25831 (version: 2011-03-08). [p. 2: The Power of Randomness: Fingerprinting and Freivalds' Algorithm-3]

[LG14] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014. [p. 2: The Power of Randomness: Fingerprinting and Freivalds' Algorithm-3]