INF561: Using randomness in algorithms **Spring 2012**

Lecture 2 — 11th of January

Lecturer: Fréderic Magniez *Scribe: KUMAR Amrit*

This lecture is meant to give an idea of the basic technique of fingerprinting and to introduce the wide spectrum of its applications and last but not least an insight to the probabilistic algorithms involved.

2.1 Hashing

A hash function takes a variable sized input message and produces a fixed-sized output. The output is usually referred to as the hash code or the hash value or the message digest.We can think of the hash code (or the message digest) as a fixed- sized fingerprint of a variable-sized message.
Hashing can be incorporated in different ways for message authentication in a communication network. A hash function can be used in conjunction with an encryption scheme and hence provides authentication in addition to confidentiality. A very different approach to the use of hashing for authentication is applied in a scheme where nothing is encrypted. However, the sender appends a secret string S, known also to the receiver, to the message before computing its hash code. Before checking the hash code of the received message for its authentication, the receiver appends the same secret string S to the message. Obviously, it would not be possible for anyone to alter such a message, even when they have access to both the original message and the overall hash code. We would be interested in the above approach all over in this lecture.

2.1.1 A Classical Example : Polynomial Identity Testing

Input : Two multivariate polynomials

$$Q(x_1, x_2 \ldots, x_n)$$
$$R(x_1, x_2 \ldots, x_n)$$

Output : If $Q = R$ then output *YES* otherwise *NO*.
If we were given both polynomials in expanded form, we could just compare coefficients. However we might get something like

$$Q(x) = \prod_{i=1}^{n-1} (x_i + x_{i+1})$$

which would take exponential time to just to expand ! Instead we do something clever :

2.1.2 Schwartz and Zippel's lemma (1979)

Let $\mathbb{F}$ be a field (finite or infinite) and S a subset of $\mathbb{F}$. Consider P a multivariate polynomial of degree d. If P is a non-zero polynomial (denoted as $P \neq 0$) over $\mathbb{F}$, which is equivalent to saying that : $\exists x_1, x_2, \dots x_n \in \mathbb{F}$ st $(P(x_1, x_2, \dots, x_n) \neq 0)$ then,

$$\mathbb{P}_{x_1,x_2\dots,x_n\in S}[P(x_1, x_2, \dots, x_n) = 0)] \leq \frac{d}{|S|}$$

Proof: We prove the above lemma by induction on n.
A non-zero polynomial in one variable of degree d has at most d roots and hence it has at most d roots in S from which we have :

$$\mathbb{P}_{x\in S}[P(x) = 0] \leq \frac{d}{|S|}$$

Supposing that the result is true for a certain $n-1$ and we write :

$$P(X_1, X_2, \dots, X_n) = \sum_{i=0}^{d} X_i^i Q_i(X_2, X_3, \dots, X_n)$$

Let j be the largest i such that $Q_i \neq 0$. We have

$$P(X_1, X_2, \dots, X_n) = \sum_{i=0}^{j} X_i^i Q_i(X_2, X_3, \dots, X_n)$$

$$\mathbb{P}_{x_1,x_2,\dots,x_n}[P(x_1, x_2, \dots, x_n) = 0] = \mathbb{P}_{x_1,x_2,\dots,x_n}[(P(x_1, x_2, \dots, x_n) = 0) \cap (Q_j(x_2, x_3, \dots, x_n) \neq 0)]$$
$$+\mathbb{P}_{x_1,x_2,\dots,x_n}[(P(x_1, x_2, \dots, x_n) = 0) \cap (Q_j(x_2, x_3, \dots, x_n) = 0)]$$

By induction hypothesis the second term T_2 of the above sum verifies :

$$T_2 \leq \frac{degree(Q_j)}{|S|}$$
$$= \frac{d-j}{|S|}$$

Fixing $x_1, x_2, \dots, x_n$ such that $Q_j(x_2, x_3, \dots, x_n) \neq 0$, we have $P(X_1, x_2, \dots, x_n)$: a polynomial in X_1 of degree j such that $P \neq 0$, for the coefficient of X_1^j is non zero. Hence we have

$$\mathbb{P}_{x_1}(P(x_1, x_2, \dots, x_n) = 0) \leq \frac{\text{degree}_{X_1}(P)}{|S|} = \frac{j}{|S|}$$

and hence the result.

□

Henceforth the subset $S = \mathbb{F} = \mathbb{Z}/p\mathbb{Z}$ where p is a prime. We suppose the following note : we can always find a prime number p such that p lies between n and $2n$ for a given n. This comes from the fact that the interval $[n, 2n[$ contains $\theta\left(\frac{n}{\log n}\right)$ primes.

Polynomial P defined as $X^3 - X$ is a non identically zero polynomial but is a zero polynomial over the field $\mathbb{Z}/3\mathbb{Z}$.

2.2 Fingerprinting and Applications

Fingerprinting

Suppose we have two copies of a large database in locations that are connected by an expensive and error-prone communication link. We wish to test the integrity of the copies, i.e. to check that both are the same. The obvious solution of sending one copy of the database across the link and doing a direct comparison is ruled out because of cost and reliability issues. In such situations, we would like to shrink our data down to a much smaller fingerprint which is easier to transmit. Of course, this is useful only if the fingerprints of different pieces of data are unlikely to be the same.

We model this situation with two spatially separated parties, Alice : A and Bob : B, each of whom holds an n-bit number (where n is very large). Alice's number is $x = x_1, x_2, \ldots, x_n$ and Bob's is $y = y_1, y_2, \ldots, y_n$. Our goal is to decide if $x = y$ without transmitting all n bits of the numbers between the parties.

Protocol

Alice picks a prime number p in the interval $[n^2, 2n^2[$ and picks randomly an integer u from the set $\{0, 1, 2, \ldots p-1\}$. These two entities are shared with Bob. Alice calculates her fingerprint as $F_{p,x}(u)$. She then sends p and $F_{p,x}(u)$ to Bob. Using p, u and y Bob computes $F_{p,y}(u)$ and checks whether $F_{p,x}(u) = F_{p,y}(u)$. If not he concludes that $x \neq y$, else he presumes that $x = y$.

Observe that if $x = y$ then Bob will always be correct. However, if $x \neq y$ then there may be an error : this happens if the fingerprints of x and y happen to coincide. We now show that, even for a modest value of p, if $x \neq y$ then $\mathbb{P}[F_{p,x}(u) = F_{p,y}(u)]$ is small.

Here we define the fingerprint function $F_{p,x}(u)$ to be

$$F_{p,x}(u) = \sum_{i=1}^{n} x_i u^i \ modulo(p)$$

Note

Each of the fingerprint obtained has $\log_2 2n^2$ bits. Each of these operations involves n multiplication and additions.

If $x \neq y$, then $(F_{p,x} - F_{p,y})[X]$ is a polynomial non identically zero. Applying Schwartz-Zeppel lemma we have :

$$\mathbb{P}_{u \in \{0,1,\ldots,p-1\}} \left[F_{p,x}(u) - F_{p,y}(u) = 0 \text{ modulo}(p)\right] \leq \frac{n}{n^2} = \frac{1}{n}$$

2.3 Applications

2.3.1 Arithmetic modulo a random prime

In many applications, we need to do arithmetic on very large integers (so large that precision is lost or overflow occurs). For example, these can occur as a result of evaluating a polynomial or a determinant. Frequently all we need to know is whether two such large integers are equal or not (e.g., does a certain polynomial evaluate to zero at some given point ?) We can deal with this problem using fingerprints in a very simple way : just do all the arithmetic modulo a random prime p. From our analysis above we know that, if the integers have n bits, then the number of bits needed in p to ensure a correct answer with high probability is only $O(logn)$, which is exponentially less and probably small enough to avoid loss of precision.

2.3.2 Pattern Matching

Problem : Let u be a pattern st $u \in \{0,1\}^m$ and a word $\omega \in \{0,1\}^n$ with $n \geq m$. The task is to find i if it exists such that u corresponds to the motif ω starting from the index i, i.e. $\omega[i]\omega[i+1]...\omega[i+m-1] = u$.

A trivial greedy algorithm consists of verifying bits at each position i. This deterministic algorithm has a complexity of $O(m.n)$. We can as well improve this algorithm using automata theory and obtain a better algorithm with complexity $O(m+n)$. We can as well desigin a probabilistic algorithm without trying to construct an automata and with the same complexity. We elucidate the later one here :

Definition 2.1. *For each word $u \in \{0,1\}^n$, we define a polynomial*

$$P_u = \sum_{i=1}^{n} u[i]X^{i-1}.$$

Theorem 2.2. *Considering $u, v \in \{0,1\}^n$, et p a prime number such that $p \geq n^3$.*
If $u = v$ then $\forall a \in \{0,1,...,p-1\}$ we have $P_u(a) = P_v(a)$ modulo (p).
If $u \neq v$ then $\mathbb{P}_{a\in\{0,1,...,p-1\}}[P_u(a) \neq P_v(a) \text{ modulo } (p)] \geq 1 - \frac{1}{n^2}$.

Proof: We considering the case where $u \neq v$. Here, $P_u \neq P_v$ ou $P_u - P_v \neq 0$.
Schwartz-Zippel's lemma gives :

$$\mathbb{P}_{a\in\{0,1,...,p-1\}}[(P_u - P_v)(a) \neq 0 \text{ modulo } p] \geq 1 - \frac{n-1}{n^3} \geq 1 - \frac{1}{n^2}.$$

□

 $P_u(a)$ mod p is the fingerprint of u.

Algorithm : Let p be a prime number between n^3 et $2n^3$.
Choose randomly $a \in \{0, 1, ..., p-1\}$, and evaluate :
- $b \leftarrow a^{m-1}$
- $f_u = P_u(a)$
- $f_v = P_{\omega[0,m-1]}(a)$
- $i \leftarrow m-1$.

do : {
if $f_u = f_v$ return i
$f_v \leftarrow \dfrac{f_v - \omega[i-m+1]}{a} + \omega[i+1] \times b$
$i \leftarrow i+1$
} while $i \leq n-1$;
Return $\emptyset$

Analysis : Complexity of the above algorithm is $O(m+n)$.
For all i, if u is a motif of ω at the position i, then the algorithm returns with the result i
If u is not a motif of ω at the position i, i.e. $u \neq \omega[i-m+1, i]$ then the algorithm return i with a probability $\leq \dfrac{n-1}{p} \leq \dfrac{1}{n^2}$.
Hence the algorithm has one-sided error. If u appears in ω, the algorithm returns the result however, if u does not appear in ω, then

$$\begin{aligned} &\mathbb{P}[\exists i \text{ st algorithm returns } [i-m+1, i]] \\ &= \mathbb{P}[\text{Algorithm returns } [1,m] \text{ or } [2,m+1] \text{ or } ...\text{or}[n-m+1, n]] \\ &\leq (m-n) \times \frac{n}{p} \sim \frac{1}{n} \text{ as } p \sim n^3 \end{aligned}$$

2.3.3 Perfect matching in bipartite graphs

Definition 2.3. *A bipartite graph $G = (A, B, E)$ is specified by two disjoint sets U and V of vertices, and a set E of edges between them. A perfect matching is a subset of the edge set E such that every vertex has exactly one edge incident on it. Since we are interested in perfect matchings in the graph G,we shall assume that $|U| = |V| = n$. The following algorithm has no error if G does not have a perfect matching (no instance), and errs with probability at most $\frac{1}{2}$ if G does have a perfect matching (yes instance).*

Theorem 2.4. *The best known deterministic algorithm for a perfect matching has a complexity of $\theta(n^{\frac{5}{2}})$.*

Theorem 2.5. *The best probabilistic algorithm with one-sided error has a complexity of $\theta(n^\omega)$, where $\omega < 2.38$. This cost corresponds to the cost of evaluating a determinant.*

Definition 2.6. *The Tutte matrix of bipartite graph $G = (A, B, E)$ is an $n \times n$ matrix F with the entry at row i and column j,*

$$F_{i,j} = \begin{cases} 0 & \text{if } (i,j) \notin E \\ X_{ij} & \text{if } (i,j) \in E \end{cases}$$

We also have :

$$Det(F) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} F_{i\sigma(i)}$$

which is a polynomial of degree at most n.

Lemma 2.7. *$Det(F) \neq 0$ iff the graph G has a perfect matching parfait.*

Proof: If $Det(F) \neq 0$ $\exists \sigma$ st $\prod_{i=1}^{n} F_{i\sigma(i)} \neq 0$ i.e. $\forall i \, (i, \sigma(i)) \in E$. The following set M is a perfect matching,

$$M = \{(i, \sigma(i))\}$$

where $i \in 1, 2, 3 \ldots n$

In the other case, if M is a perfect matching, then considering $\forall i$ $\sigma(i) = j$ st $(i, j) \in M$ where $\sigma \in S_n$ because M is a perfect matching. This entails that $\prod_{i=1}^{n} F_{i\sigma(i)} \neq 0$ and hence $Det(F) \neq 0$ because of the unique decomposition of $Det(F)$.

□

Algorithm :

Let p be a prime, st $n^2 < p < 2n^2$.
- Choose randomly $x_{ij} \in \{0, 1, ...p-1\}$.
- Evaluate $d = Det(F)(x_{ij})$ modulo p, and substitute X_{ij} by x_{ij} in F.
- Return ACCEPT if $d \neq 0$.
- Else return REJECT .

Remark

- The complexity of the problem corresponds to the evaluation of a determinant.
- According to the above lemma, if the graph has no perfect matching, the algorithm always returns REJECT.
- If it exists : We prove that the algorithm returns ACCEPT with a probability greater than $1 - \frac{1}{n}$.
  This is a direct consequence of the fact that $Det(F)$ is a polynomial of degree less than n, with at most n^2 variables and and the Schwartz-Zippel's lemma .

2.3.4 Associativity Testing

Input : Let S be a set of n elements for example $S = \{1, 2, \ldots, n\}$ and a binary operation $\circ$ on S.
Output : Decide if $\forall i, j, k \; (i \circ j) \circ k = i \circ (j \circ k)$

Definition 2.8. *For $p \geq 7$, a prime number and $A, B, C \in (Z_p)^n$ define a binary operation $\bullet$ over $(Z_p)^n$ st $(A \bullet B)_k = \sum_{i,j \text{ st } i \circ j = k} A_i \times B_j$*

Let $e_i = (0, 0, \ldots, 1, 0, 0 \ldots, 0)$ where 1 is at the ith position. Then we have :

$$A = \sum A_i \vec{e_i}$$
$$B = \sum B_j \vec{e_j}$$

Remark

– We also have : $\vec{e_i} \bullet \vec{e_j} = \vec{e}_{i \circ j}$ Its bilinear extension becomes :

$$A \bullet B = \left(\sum_{i \in S} A_i \vec{e_i} \right) \bullet \left(\sum_{j \in S} B_j \vec{e_j} \right)$$
$$= \sum_{i,j \in S} A_i A_j \vec{e}_{i \circ j}$$

– Associativity of $\circ$ on $S \Leftrightarrow$ associativity of $\bullet$ over $(Z_p)^n$

– If $\circ$ is not associative, then :

$$\mathbb{P}_{A,B,C \in (Z_p)^n}[(A \bullet B) \bullet C = A \bullet (B \bullet C)] \leq \frac{3}{p}$$

The first remark about the associativity of $\bullet$ becomes evident by expanding the product using the basis vectors. For the second remark : fix $r \in S$ and calculate :

$$\Delta_r(A, B, C) = (A \bullet B) \bullet C - A \bullet (B \bullet C)$$
$$\Delta_r(A, B, C) = \sum_{i,j,k \ (i \circ j) \circ k = r} A_i B_j C_k - \sum_{i,j,k \ i \circ (j \circ k) = r} A_i B_j C_k$$

which is a polynomial in $(A_i B_j C_k)$ of degree at most 3. If $\circ$ is not associative, then $\bullet$ neither. Hence, $\exists r_0$ st $\Delta_{r_0} \neq 0$. Concluding, by Schwartz-Zippel's lemma we obtain the result.

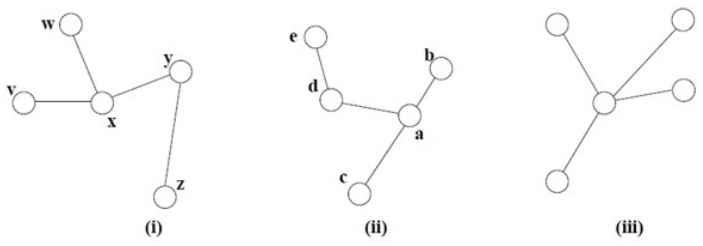
Using the last remark, we can easily design a probabilistic algorithm which can solve the above problem of associativity testing. The complexity of this algorithm will be reduced to the complexity of evaluating the following product :

$$A \bullet B = \left(\sum_{i \in S} A_i \vec{e_i} \right) \bullet \left(\sum_{j \in S} B_j \vec{e_j} \right)$$

which is of order $O(n^2)$. So, verifying the expression $(A \bullet B)C = A \bullet (B \bullet C)$ entails a complexity of $O(4n^2)$.

2.3.5 Tree Isomorphism

Definition 2.9. *Informally, we say that two graphs are isomorphic if one can be transformed into the other simply by renaming nodes. For example, in the illustration below, the first two trees are isomorphic but the third is not isomorphic to either of the other two trees. Formally, we say that two graphs $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are isomorphic if there is a 1-1 mapping $f : V_1 \to V_2$ such that (v, w) is an edge in E_1 iff $(f(v), f(w))$ is an edge in E_2. We call such a mapping a tree isomorphism.*



Problem :

Let T_1, T_2 be unordered trees of height h and with n nodes. Decide if T_1 et T_2 are isomorphic. We look for an algorithm with time complexity $O(n)$ and space complexity $O(n \times logn)$.

Definition 2.10. *For a tree T with root u, define inductively a polynomial $f_{T,u}$ corresponding to T as follows :*
- If T has just one node u, then $f_{T,u} = X_0$
- If the height of the tree is $h \geq 1$ and T has k subtrees $T_i, 1 \leq i \leq k$ with roots u_i, then

$$f_{T,u} = \prod_{i=1}^{k} (X_h - f_{T_i,u_i})$$

Theorem 2.11. *Trees T_1, T_2 with roots u_1, u_2 are isomorphic iff*

$$f_{T_1,u_1} = f_{T_2,u_2}$$

Proof: Using the definition of the polynomial used as a fingerprint, the two trees are isomorphic, iff the polynomial f corresponding to them are identical. However, let $f_{T_1,u_1} = f_{T_2,u_2}$. By construction, the degree d of these two polynomials is equal to one more than the height h of the trees. We prove by induction on h that the two trees are isomorphic, where the height of the tress are same.
For $h = 0$, the proposition is valid.
Suppose that the proposition is valid for all height $< h$. Consider $f_{T_1,u_1} = f_{T_2,u_2}$ as a polynomial in X_h. Since, this polynomial has a unique factorization in monomials in X_h ; T_1 et T_2 must have the same number of subtrees (i.e. k) and if $T_{i,j}$ are the sub-trees of T_i, then there exits a permutation π of $\{1, 2, ..., k\}$ st

$$\forall i = 1, 2, ..., k, f_{T_{1,i},u_{1,i}} = f_{T_{2,\pi(i)},u_{2,\pi(i)}}$$

Using induction hypothesis, the subtree $T_{1,i}$ is isomorphic to the subtree $T_{2,\pi(i)}$ for all i, et hence T_1 is isomorphic to T_2. □

Algorithm : Using the construction above we can assign a polynomial $f_{T,u}$ to each tree T and the isomorphism of the trees is characterized by the fact that they have the same polynomial representation. Application of Schwartz-Zippel's lemma assures it to be a probabilistic algorithm with one-sided error.

2.4 Probabilistic algorithm for satisfiability : SAT

Definition 2.12. *Let $X_1, X_2, ..., X_n$ be $n \geq 1$ logic variables. A literal l is of the form X_i or $\overline{X_i}$ i.e. a literal is either a variable or the negation of a variable*
A clause C is a disjunction of literals, ex : $C = X_1 \vee X_2 \vee \overline{X_3}$
A clause C containing at most k variables is also called a k-clause. A SAT formula θ is of the form $\theta = C_1 \wedge C_2 \wedge ... \wedge C_m$ where C_i are clauses. A SAT formula θ is called k-SAT formula if θ contains only k-clauses.
The k-SAT is a decision problem to decide if for a given k-SAT formula, there exists a valuation for the boolean variables for which the formula is true.

Facts :

The following theorems have been given without their proof.

Theorem 2.13. *2-SAT $\in P$*
k-SAT is NP-complete for all $k \geq 3$

Theorem 2.14. *2-SAT can be solved by a deterministic algorithm with complexity $O(n+m)$ where n is the number of variables and m is the number of clauses.*

We are interested in designing a probabilistic algorithm to solve k-SAT and which works for all k.

Algorithm

- Choose randomly an $a \in \{0,1\}^n$
- Repeat until $\theta(a) \neq 1$ (and at most $2kn^2$ iterations for the variant)
  Choose arbitrarily a clause C_j st $C_j(a) = 0$
  Flip the value of one of the variables in C_j
  Update a
- If $\theta(a) = 1$ accept else reject.

Remarks

- If $\forall a, \theta(a) = 0$, algorithm (with no variant) does not terminate.
- If the algorithm (with no variant) stops then $\theta(a) = 1$
- One iteration has a complexity of $O(mk)$.

Theorem 2.15. *If θ is 2-SAT and that it has a solution a st $\theta(a) = 1$ then the average number of iteration needed is $\leq n^2$.*

Corollary 2.16. *If $\forall a \theta(a) = 0$ then the algorithm always rejects.*
If $\exists a$ st $\theta(a) = 1$ then $\mathbb{P}[\text{Algo rejects}] \leq \frac{1}{2}$. This is a direct consequence of the above theorem and Markov's Inequality for a positive random variable T and any real $\mu \geq 0$:

$$Pr[T \geq \mu] \leq \frac{1}{\mu}(E(T))$$

An extension to the above corollary is also true for a k-SAT where the probability of rejection becomes $(\frac{1}{2})^k$.

When considering the variant that loops at most $2k^n 2$, an execution with a total of $2kn^2$ iterations behaves as k independent runs whose each execution iterates $2n^2$.