

## Lecture 1 — 4th January 2012

Lecturer: Frédéric Magniez

Scribe: Arindam Biswas

The goal of this course is to furnish the definition of different models and also to present some examples of probabilistic algorithms.

## 1.1 Randomness in Computation and Randomized Algorithms :

A randomized algorithm is an algorithm that in addition to taking the usual input also takes a string of independent, uniformly distributed random bits. We may then view the (randomized) algorithm as a traversal of a decision tree, where we take the left branch at each stage if the next random bit is 0 and the right branch if it is 1. If the tree has height  $k$ , then each possible outcome has probability  $2^{-k}$ . The above is a strict model. In practice, we can say things like, "with probability  $2/3$  do..." because we can simulate any such probability with small error. We can also say "choose a random  $x \in [0, 1]$ ". The error of these approximations can be added to the error inherent in our algorithms as long as our algorithm does not depend in very precise ways on the probabilities.

## 1.2 Model

We consider accessible the randomly generated numbers (random resources)  $r \in \{0, 1\}^*$ . From here we can easily consider accessible  $r \in [[a, b]]^*$  where  $a$  and  $b$  are integers.

**Note :** We do not yet know how to generate the random numbers (which is a question important and difficult).

Let  $f : X \rightarrow Y$  and  $A$  an algorithm (Here  $X$  and  $Y$  are finite sets).

**Definition 1.1.** We say that  $A$  calculates  $f$  without an error with an average complexity  $T$  if and only if :

1.  $\forall x \in X, A(x)$  outputs  $f(x)$
2.  $\forall x \in X, E_{r \in \{0,1\}^*} (C(A(x,r))) \leq T(|x|)$  where  $|x|$  is the size of the entry and  $C(A(x,r))$  is the complexity of  $A$  over the entry  $x$  with the random choices  $r$ .

We study the complexity on average.

e.g; Quicksort with random pivot.

**Definition 1.2.** We say that  $A$  calculates  $f$  without error with a probability of failure  $\delta$  and a complexity  $T$  if and only if :

1.  $\forall x, \forall r$ , if  $A(x, r)$  does not abort, then  $A(x, r)$  gives  $f(x)$
2.  $\forall x, \Pr(A(x, r) \text{ aborts}) \leq \delta$
3.  $\forall x, \forall r, C(A(x, r)) \leq T$  (worst case)

In here we study the complexity in the worst case.

⚡ In general we choose  $\delta = 1/2$ . To pass from  $\delta_0 = 1/2$  to  $\delta_k = 1/2^k$  we iterate  $k$  times the same algorithm with the new random bits and we get  $T(\delta = 1/2^k) \leq kT(\delta = 1/2)$ . Suppose  $\delta_k = 100$  then we need to iterate  $\lceil \log_2 100 \rceil = 7$  times.

⚡ All algorithms of the previous form can be converted to this form. It is sufficient to stop the algorithm if  $T$  becomes too large.

e.g; Quicksort with random pivot and finite time of execution.

⚡ From now on we consider the languages :  $Y = \{0, 1\}^*$  and  $L = \{x \mid f(x) = 1\}$ .

**Definition 1.3.** One sided error (RP)

$A$  calculates  $f$  with a one-sided error  $\delta$  and a complexity  $T$  if and only if :

1.  $\forall x, \forall r, C(A(x, r)) \leq T$
2. if  $x \in L$ , then  $\Pr(A(x, r) \text{ accepts}) = 1$
3. if  $x \notin L$ , then  $\Pr(A(x, r) \text{ accepts}) \leq \delta$

⚡ In general we choose  $\delta = 1/2$ . (it is sufficient to have  $\delta < 1$ ).

⚡ Similar to the above, if we wish to have  $\delta = 1/2^k$ , we reiterate the algorithm  $k$  times and accept the result if and only if the  $k$  executions are all accepted.

**Definition 1.4.** Co-RP

$A$  is in Co-RP if and only if :

1.  $\forall x, \forall r, C(A(x, r)) \leq T$
2. if  $x \in L$ , then  $\Pr(A(x, r) \text{ accepts}) \geq 1 - \delta$
3. if  $x \notin L$ , then  $\Pr(A(x, r) \text{ accepts}) = 0$

**Definition 1.5.** Two-sided error (BPP)

$A$  calculates  $f$  with a two-sided error  $\delta$  and a complexity  $T$  if and only if :


1.  $\forall x, \forall r, C(A(x, r)) \leq T$
2. if  $x \in L$ , then  $\Pr(A(x, r) \text{ accepts}) \geq 1 - \delta$
3. if  $x \notin L$ , then  $\Pr(A(x, r) \text{ accepts}) \leq \delta$

⚡ In this case without loss of generality we can take  $\delta < \frac{1}{2}$ . In general we choose  $\delta = \frac{1}{3}$ . In the last case we accept the results if the majority of the results are accepted.

## 1.3 First Examples :

### Example 1 : Primality

$$L = \{p | p \text{ nombre premier}\}, x \in \mathbb{N}^*, |x| = \log_2(x).$$

 The sieve of Eratosthenes gives us a result in  $\sqrt{n}$  steps which is too long.

**Theorem 1.6.** *Rappel : Fermat's Little Theorem*

$$p \text{ prime number} \implies \forall a \in \{1, 2, \dots, p-1\}, a^{p-1} \equiv 1 \pmod{p}$$

Two simple methods to prove the above result are :

a) We proceed by induction over  $a$ , namely we assume the original version of the theorem i.e.  $a \in \mathbb{N}$  and show that if the result holds for  $a = k$  then it also holds for  $a = k + 1$ .

b) We prove using the theory of groups for which we recognise that the set  $G = \{1, 2, \dots, p-1\}$ , with the operation of multiplication (taken modulo  $p$ ), forms a group. Then if  $k$  denotes the order of  $a$  we have  $a^k \equiv 1 \pmod{p}$ . We then apply Lagrange's theorem (namely the order of every element divides the order of the group which is  $p-1$  in this case) to get the result.

**Algorithm :**

1. We choose  $a$  at random, uniformly from the set  $\{1, 2, \dots, p-1\}$ .
2. If  $a \wedge p \neq 1$  we reject the result.
3. If  $a^{p-1} \not\equiv 1 \pmod{p}$  we reject the result.
4. If not we accept the result.

 The complexity of this algorithm is logarithmic in the number of arithmetic operations.

**Theorem 1.7.** *The previous algorithm is of type One-sided error.*

**Proof:** – If  $p$  is prime, the algorithm accepts each time (Fermat's Little Theorem). If  $p$  is not prime then we have 2 cases (namely because the statement of Fermat's Little Theorem is not if and only if)

– We recall the fact that a number  $p$  (not prime) is called a Carmichael number if for all  $a$  relatively prime to  $p$  we have  $a^{p-1} \equiv 1 \pmod{p}$ .

Suppose that  $p$  is not a Carmichael number.

Let  $a_0$  such that  $a_0 \in \{0, 1, \dots, p-1\}$ ,  $a_0 \wedge p = 1$  and  $a_0^{p-1} \not\equiv 1 \pmod{p}$ .

**Lemma 1.8.** *Pr  $a \in \{0, 1, \dots, p-1\}$  et  $a \wedge p = 1$   $[a^{p-1} \equiv 1 \pmod{p}] \leq \frac{1}{2}$*

**Proof:** Let  $G$  be the multiplicative group such that  $G = \{a | a \wedge p = 1\}$  et  $H$  le subgroup de  $G$  such that  $H = \{a \in G | a^{p-1} \equiv 1 [p]\}$ .

$a_0 \notin H, 1 \in H$  et  $a_0 \in G$  and so  $H$  is a non-empty strict subgroup of  $G$  and hence using Lagrange's theorem we get  $|H| \leq \frac{|G|}{2}$ .

□

The lemma implies that if  $p$  is neither prime nor a Carmichael number we have :

$$Pr_a [the algorithm rejects in (3) knowing that it accepted in (2)] \geq \frac{1}{2}$$

We recall that  $Pr(A \cap B) = Pr(B)Pr(A|B)$ .

$$\begin{aligned} & Pr_a [the algorithm rejects in (2) or in (3)] \\ &= Pr_a [the algorithm rejects in (2)] + Pr_a [the algorithm rejects in (3) and not in (2)] \\ &= \Delta + (1 - \Delta)Pr_a [the algorithm rejects in (3) | the algorithm accepts in (2)] \\ &\geq \Delta + \frac{1-\Delta}{2} = \frac{1+\Delta}{2} \geq \frac{1}{2} \end{aligned}$$

$p$  is now a Carmichael number

$$p - 1 = 2^t u$$

We wish to calculate  $a^{p-1} [p]$ . Let  $i$  be the last integer such that  $v = a^{2^i u} \neq 1 [p]$   
 $v^2 = 1 = a^{2^{i+1} u}$ . (if  $p$  is prime we have  $v = -1$ )

**Theorem 1.9.**  $Pr_a [v = -1 \text{ or } i \text{ does not exist}] \leq \frac{1}{4}$



For details see paper of Miller and Robin.

□

### Utility and examples of Cryptographic codes :

Now we ask the question that what is the utility of the above? One of the main applications is in Cryptography which is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). Modern cryptography is mainly based on the fact that in the theory of computation there exist problems which are easy to compute but the inverse problems are much harder for computers to calculate.

e.g. - Given two sufficiently large primes  $p$  and  $q$ , with the help of computers we can easily calculate their product  $n = pq$  however if we are supplied  $n$  and we wish to decompose it into prime factors then the task takes relatively longer to accomplish.

Examples of Cryptographic Codes are :

a) Diffie-Hellmann Key Exchange

b) RSA Cryptography

etc. They are both examples of Public-Key Cryptography or more generally known as Asymmetric Key Cryptography (one in which two different, but mathematically related keys are used. One for encryption and the other for decryption of a message). This has considerable advantage over symmetric key crypto-systems.

We are going to speak a little about the RSA encryption.

One thing to note is that in both of the above, everything really depends on generating

randomly a sufficiently large prime number.

How to achieve this? We take a random number and see whether it is prime or not.

How to do that? → use Primality Test.

### RSA Encryption Method :

The RSA algorithm was publicly described in 1978 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT; the letters RSA stand for initials of their surnames.

How does it work?

#### Coding :

Choose two distinct sufficiently large primes  $p$  and  $q$ .

Let  $n = pq$  denote their product called modulus of coding.

Calculate Euler's phi function of  $n$ ,  $\phi(n) = (p-1)(q-1)$ .

Choose  $e$  an integer relatively prime to  $\phi(n)$  called exponent of coding

As  $e$  is prime to  $\phi(n)$ , so from Bézout's theorem we get that it is invertible modulo  $\phi(n)$  i.e. there exist an integer  $d$  such that  $ed \equiv 1 \pmod{\phi(n)}$ .  $d$  is called the exponent of decoding.

The couple  $(n, e)$  is called public key while the couple  $(n, d)$  is called private key.

If  $M$  is an integer less than  $n$  representing a message, then the coded message is represented by  $C$  where  $C \equiv M^e \pmod{n}$

#### Decoding :

For decoding  $C$ , we use  $d$ , the inverse of  $e$  modulo  $\phi(n)$  and we calculate  $C^d \pmod{n}$

We have,  $C^d \pmod{n} \equiv (M^e)^d \pmod{n} \equiv M^{ed} \pmod{n}$

As  $ed \equiv 1 \pmod{\phi(n)}$  by definition of modulo we have  $ed = 1 + k\phi(n) = 1 + k(p-1)(q-1)$  with  $k \in \mathbb{N}$ .

Or for all integers  $M$ ,  $M^{1+k(p-1)(q-1)} \equiv M \pmod{p}$  and  $M^{1+k(p-1)(q-1)} \equiv M \pmod{q}$  (using Fermat's little theorem)

which in turn gives  $M^{1+k(p-1)(q-1)} \equiv M \pmod{pq}$  as  $p \wedge q = 1$ .

So we have  $C^d \equiv M^{ed} \equiv M^{1+k(p-1)(q-1)} \equiv M \pmod{n}$ .

This solves the problem. We reiterate that for coding, it is sufficient to know  $e$  and  $n$ , while for decoding we need  $d$  and  $n$ . To calculate  $d$  with the help of  $e$  and  $n$ , we need to find the inverse of  $e$  modulo  $(p-1)(q-1)$  which forces us to know the primes  $p$  and  $q$  i.e. the decomposition of  $n$  into prime factors.

### Example 2 : Verification of a matrix product

#### Problem :

Consider three  $n \times n$  input matrices  $A$ ,  $B$  and  $C$  with integer entries. The algorithm ACCEPTS and returns 1 if  $C$  is the matricel product of  $A$  and  $B$  else it REJECTS and returns 0.

#### Freivald's Algorithm :

1. Choose  $r \in \{0, 1\}^n$ .
2. Evaluate  $u = Cr$ ,  $v = Br$  and  $w = Av$ .

3. Return ACCEPT if  $u = w$ , else REJECT.

**Theorem 1.10.** *Freivald's Algorithm has one-sided error : the decision problem responds YES, then the algorithm accepts it as well, while in the other case, the algorithm accepts it with a probability  $Pr[ACCEPT] \leq \frac{1}{2}$  and with a time complexity  $\theta(n^2)$ .*

**Proof:**

- Number of multiplications (matrix with a vector) comes out to three and each has a complexity of order  $n^2$ .
- If  $AB = C$  algorithm returns ACCEPT.
- Let  $AB \neq C$ , we prove that  $\exists r \in \{0, 1\}^n$  such that  $ABr \neq Cr$ . Consider the basis vectors of the vector space  $\{0, 1\}^n$ . If  $(AB)_{ij} \neq (C)_{ij}$ , then taking  $r = (0, 0, 0 \dots 1 \dots 0)$  with 1 at the  $j$ 'th position serves our purpose.
- Considering the case where the matrix entries are  $\{0, 1\}$ . Define a set  $S = \{r | (AB - C)r = 0\}$

$$S = Ker(AB - C)$$

such that  $S$  is a proper subspace of the  $n$ -dimensional vector space over the field  $Z/2Z$ . Hence  $S$  forms a subgroup of the additive group of the field  $Z/2Z$  and so satisfies :

$$|S| \leq \frac{|\{0, 1\}^n|}{2}.$$

Hence the result. □

**Theorem 1.11.** *In the general case where the matrix entries form a ring.*

**Proof:** Define  $D = AB - C$ .

Let  $D_{i_0 j_0}$  be a non-zero entry of  $D$ .  $\forall r$ ,  $(Dr)_{i_0} = \sum_{j=1}^n D_{i_0 j} r_j = D_{i_0 j_0} r_{j_0} + \sum_{j \neq j_0} D_{i_0 j} r_j$

We fix  $(r_1, r_2, \dots, r_{j_0-1}, r_{j_0+1}, \dots, r_n)$ .

$r^0 = (r_1, r_2, \dots, r_{j_0-1}, 0, r_{j_0+1}, \dots, r_n)$  and  $r^1 = (r_1, r_2, \dots, r_{j_0-1}, 1, r_{j_0+1}, \dots, r_n)$ .

As  $D_{i_0 j_0} \neq 0$  we have  $(Dr^0)_{i_0} \neq (Dr^1)_{i_0}$  and this concludes that the algorithm holds in the general case as well. □

### Example 3 : Test of commutativity

**Problem :**

Having given a finite group  $G$  and  $n$  elements  $h_1, h_2 \dots h_n$  of  $G$ , we wish to find an algorithm which accepts if  $\forall (i, j)$ ,  $h_i h_j = h_j h_i$  and rejects if not.

We are interested in the complexity in number of group operations (and not on the number of additions, multiplications... even though  $G$  can be  $M_n$  for example).

We recall that the center of the group  $G$  is defined by :  $Z(G) = \{k \in G | \forall g \in G, kg = gk\}$  and  $H$  is the sub-group generated by the  $h_i$ .



We do not know how to calculate efficiently.

**Hypothesis :**

We suppose that we know how to generate uniformly at random an element of  $H$ .

**Algorithm :**

- Generate  $h$  and  $k$  uniformly at random in  $H$ .
- Output ACCEPT if  $hk = kh$ .
- Output REJECT if not.

**Theorem 1.12.** *This is an algorithm of type One-sided error of complexity 2 operations of group and 2 samples in  $H$ .*

**Proof:** – The complexity follows.

- If the  $h_i$  commute the algorithm outputs ACCEPTER.
- Suppose that  $H$  is non-commutative.  $Z(H) \neq H$  and  $H$  is a sub-group and hence  $Pr_{h \in H} [h \in Z(H)] \leq \frac{1}{2}$ .

We note that  $L_h = \{g \in H | gh = hg\}$ . We have  $L_h \neq H$  if  $h \notin Z(H)$ .

Let  $h \notin Z(H)$ .

As  $L_h$  is a strict sub-group of  $H$  we have :  $Pr_{k \in H} [k \in L_h] \leq \frac{1}{2}$ .

Finally,,  $Pr_{k,h} [kh \neq hk] \geq Pr [h \notin Z(H) \text{ and } k \notin L_h] =$

$$Pr [k \in L_h | h \notin Z(H)] \times Pr [h \notin Z(H)] \geq \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$$

□

**Problem of generation of elements at random in H :**

The uniform generation at random of elements in  $H$  is a difficult question. Even then to get the result it is sufficient to have a generator of elements of  $H$  such that if  $K$  is a strict sub-group of  $H$ ,  $Pr_h \text{ coming from our generator } [h \in K] \leq \frac{1}{2}$  ( $\diamond$ )

**Generator :**

- Choose uniformly at random  $r \in \{0, 1\}^n$ .
- Calculate  $h = h_1^{r_1} h_2^{r_2} \dots h_n^{r_n}$
- Output  $h$ .

**Lemma 1.13.** *The generator satisfies ( $\diamond$ ).*

**Proof:**  $K$  is a strict sub-group of  $H$ .

Let  $i_0$  be such that  $h_{i_0} \notin K$ .  $r \in \{0, 1\}^n$ .

$$h = h_1^{r_1} h_2^{r_2} \dots h_{i_0}^{r_{i_0}} \dots h_n^{r_n}$$

We fix  $r_i$  for  $i \neq i_0$ . We note  $h^0$  the product for  $r_{i_0} = 0$  et  $h^1$  for  $r_{i_0} = 1$ .

$$h^0 = ab \text{ and } h^1 = ah_{i_0}b.$$

We choose  $i_0 = \min \{i | h_i \notin K\}$  in such a way that it is in  $K$ . If  $b \in K$  then  $h^0 \in K$  but  $h^1 \notin K$  as otherwise  $a^{-1}h^1b^{-1} = h_{i_0} \in K$  which is false according to our hypothesis.

If  $b \notin K$  we have directly  $h^0 \notin K$ . We see that at least one of the 2 elements is not in  $K$  and we conclude. □