A - II - ③ Computational Complexity

a) Notions from classical computer science

We saw previously that (deterministic) classical computations can be reduced to the evaluation of Boolean fcts :

$$\longrightarrow \{0,1\}^n \longrightarrow \{0,1\} .$$

which can be decomposed into a sequence of logical operations (classical gates) such as NOT, AND, OR ...

In fact the set of gates {NOT, AND, OR} is said to be "universal" because any fct can be evaluated by building a circuit from these components only.

Note : ∃ other universal sets of gates such as {AND, XOR}, {NAND} ...

A Boolean fct may be said to encode a solution to a "decision pb", i.e. the fct examines the input and issues a YES (1) or NO (0) answer.

In the following, we will restrict ourselves to such decision pbs.

Ex: * "Primality pb"
↳ is integer p a prime number?

* "Factoring pb"
↳ determine the prime factors of an integer:

$$p = q_1\, q_2\, q_3 \ldots \qquad (q_i = \text{prime})$$

not a priori a decision pb but can be recasted into one:

$$f(p,m) = \begin{cases} 1 \text{ if } p \text{ has a divisor } q \\ \quad \text{with } 1<q<m \ (<p) \\ 0 \quad \text{otherwise} \end{cases}$$

$f(p,m)$: 2 integers

Example : $p = 30$.

- $f(30, 3) \longrightarrow$ yes : $q_1 = 2 < 3$ divides $p = 30$.
  $\Rightarrow q_2 = \frac{30}{2} = 15$ not prime
- $f(15, 3) \longrightarrow$ no. ($\nexists 1 < q_1 < 3$ which divides 15)
- $f(15, 5) \longrightarrow$ yes : $q_1' = 3 < 5$ divides 15
  $\Rightarrow q_2' = 15/5 = 5$ prime $\rightarrow$ stop.

$$\Rightarrow p = q_1 q_1' q_2'$$
$$30 = 2 \times 3 \times 5 \quad \checkmark$$

In these pbs, the size of the input is the number of bits needed to represent the integers p and n.

$\Rightarrow$ Each decision pb really defines a family of Boolean fcts with variable input size.

We will denote such a fct family as

$$f : \{0,1\}^* \longmapsto \{0,1\}$$

where the star indicates that the input size is variable.

The set of strings "accepted" by a fct family
↳ with output 1

$$L = \{ x \in \{0,1\}^* \text{ such that } f(x) = 1 \}$$

is called a language.

⇒ the decision pb is to determine whether a given string is in L or not.

In general, when we consider a given computational pb, one important question is "what are the minimal resources (e.g. time, memory) required to solve this pb ?"

In the circuit model, it is natural to use the size of the circuit
(= number of gates)
to characterize the required resources.

To be more precise, we can also define:

the "depth" of a circuit
= # of time steps required,
assuming that gates that act on
≠ bits can be executed simultaneously.

Example:

⇒ depth = 4

This is directly related to the time of the computation.

Typically we will be interested in quantifying how the size of a circuit of a decision pb (with variable input size) depends on the size n of the input.

→ we are not interested in the exact # of gates, but rather how this number scales with n.

In computational complexity, one makes a distinction btw

* problems that can be solved using circuits with a size that is $O(\text{poly}(n))$

$$\text{size(circuit)} \leq \underbrace{\text{poly}(n)}_{\hookrightarrow \text{some polynomial fct of } n.} \quad (\text{for large } n)$$

→ the circuit is said to be "polynomial size"

⇒ "EASY" PROBLEMS.

* problems that require circuit sizes which grow faster that any polynomial in n

→ the circuit is said to be "exponential size"

⇒ "HARD" PROBLEMS.

* Notes:

- we call "exponential" everything that is not polynomial - (even $n^{\log n}$ which is in pcple in btw)
- whether a pb is "easy" or "hard" does not depend on what universal gate set we choose to build the circuit.

⇒ one universal gate set can "simulate" another one "efficiently".

* Primality pb → easy.
(proven in 2002, AKS)

* Factoring pb is <u>believed</u> to be hard
(so far no algo has been found to solve the factoring pb with poly-size classical circuits)
→ but not proven.

In general, it is very difficult to <u>prove</u> that some particular pbs require expo. resources to solve.

(it is typically much easier to assess upper bounds to the number of required gates, than lower bounds)

⇒ <u>consequence</u>: until such proofs are established, it cannot be concluded with certainty that a quantum computer will be more powerful than a classical one.
(although it is strongly believed)

* Complexity classes

A cplexity class = set of decision pbs that all have similar cplexity.

① "P" ("polynomial time")

= class of pbs that are "easy", i.e. that can be solved by poly-size circuits.

factoring is believed $\notin$ P.

② "NP" ("non-deterministic poly-time")

= class of pbs for which the "yes" instances can be easily checked (with poly-size circuits) with the aid an appropriate witness _

Rigorous definition of NP: (can be skipped)

A language $L \in NP$ if $\exists$ a poly-size "verifier" $V(x, y)$ such that

- if $x \in L$, $\exists\, y$ such that $V(x, y) = 1$ "completeness"
- if $x \notin L$, $\forall\, y$ $V(x, y) = 0$ "soundness"

The verifier is the circuit that checks the answer.

"completeness" → for each input $\in L$, $\exists$ a witness such that the verifier accepts the input if the witness is provided.

"soundness" → for each input $x \notin L$, the verifier will reject the input no matter what witness is provided.

Notes : * Clearly $P \subseteq NP$

* Fundamental Conjecture :

$$P \neq NP$$

* NP-completeness : a pb A is "NP-complete" if any other pb in NP is reducible to A.

⇒ the NP-complete pbs are the hardest because if we can solve any of them then we can solve any pb in NP.

③ "co-NP" (complementary of NP)

= class of pbs that have witnesses to "no" instances _

(can be answered by exhibiting a counter-example).

Rigorous definition : $L \in$ co-NP if $\exists$ poly-size verifier $V(x,y)$ such that

$$\begin{cases} \text{if } x \notin L \Rightarrow \exists\, y \text{ such that } V(x,y) = 1 \\ \text{if } x \in L \Rightarrow \forall\, y \; V(x,y) = 0 \end{cases}$$

Note: factoring pb is both in NP and co-NP.

In addition to time classification, we can also organize pbs in terms of how much space they require
↳ storage space, memory

⇒ PSPACE = set of pbs that can be solved with space that ↑ polynomially with n.

(no restriction on time)

So far we have considered deterministic classical computations (the output is fully determined by the input) involving irreversible gates (such as AND gate).

There a couple of aspects that can be introduced into classical computation, to take us closer to quantum case. These are 1) randomness and
2) reversibility.

* Randomized classical computations

It is possible to add randomness to classical computations by providing the computer access to a random-number generator whose results decide what actions are taken during the computation.

=> even if the input is fixed, the circuit samples from many possible computational paths -

An algorithm performed by such probabilistic circuit is said to be "randomized".

If we run a randomized computation many times on the same input, we will not get the same answer every time, instead there is a proba. distribution of outputs.

A randomized computation will be useful if the proba. of getting the right answer is high enough, i.e. if

$$\text{proba}(\text{right answer}) \geq \frac{1}{2} + \delta$$

where $\delta > 0$ is a constant independent of the input size.

In the case of a decision pb, we want the randomized computation to accept an input $x \in L$ (L = language) with proba $\frac{1}{2} + \delta$, and to accept input $x \notin L$ with proba $\frac{1}{2} - \delta$.
↳ wrong answer

⇒ Then we can "amplify" the proba of success by performing the computation many times and taking the majority vote of the outcome.

In fact, for $x \in L$, if we run the computation N times, the proba of rejecting x (i.e. of getting the wrong answer) in more than half of the runs is no more than $\exp(-2N\delta^2)$.
↳ "Chernoff bound".

Similarly, the proba of accepting $x \notin L$ in more than half of the runs is no more than $\exp(-2N\delta^2)$.

Proof : If we do N trials, there are 2^N possible sequences of outcomes (1 outcome is one bit 0 or 1)

and the proba of any particular sequence with N_W wrong answers is

$$\left(\frac{1}{2}-\delta\right)^{N_W} \left(\frac{1}{2}+\delta\right)^{N-N_W}$$

$\left(\frac{1}{2}-\delta\right)$: proba of getting wrong answer once in the sequence

Now the majority (in the partic. sequence of N trials) is wrong if $N_W \geq N/2$.

In that case :

$$\left(\frac{1}{2}-\delta\right)^{N_W} \left(\frac{1}{2}+\delta\right)^{N-N_W} \leq \underbrace{\left(\frac{1}{2}-\delta\right)^{N/2} \left(\frac{1}{2}+\delta\right)^{N/2}}$$

$$\frac{1}{2^{N/2}}\left(1-2\delta\right)^{N/2} \frac{1}{2^{N/2}}\left(1+2\delta\right)^{N/2}$$

$$= \frac{1}{2^N} \left(1 - 4\delta^2\right)^{N/2}$$

$(1-x) \leq e^{-x}$

$$\leq \frac{1}{2^N} \left(e^{-4\delta^2}\right)^{N/2} = \frac{e^{-2N\delta^2}}{2^N}$$

This was for one particular sequence

$\Rightarrow$ to get the total proba of getting the wrong answer the majority of the time, we simply multiply by the number of sequences $= 2^N$.

$\Rightarrow$ proba (wrong majority) $\leq e^{-2N\delta^2}$

$\Rightarrow$ drops exponentially as the number of runs increases.

Thus the proba of error is less than a given ε if we run the computation at least $N = \frac{\ln(1/\varepsilon)}{2\delta^2}$ times

$$e^{-2N\delta^2} \leq \varepsilon \Rightarrow -2N\delta^2 \leq \ln \varepsilon$$

$$\Rightarrow N \geq \frac{-\ln(\varepsilon)}{2\delta^2} = \frac{\ln(1/\varepsilon)}{2\delta^2}.$$

Notes :

* N logarithmic in $1/\varepsilon$ → can make the error ε very small by repeating the computation a modest # of times.

* δ independent on the size of the input
  ⇒ so is N.

  it only depends on the error ε that we are willing to make and on the value of δ which we also decide.

The standard convention is to take $\delta = 1/6$.

⇒ $\left.\begin{array}{l} x \in L \text{ is accepted} \\ x \notin L \text{ is rejected} \end{array}\right\}$ with proba $\geq 2/3$.

$\left.\begin{array}{l} x \in L \text{ is rejected} \\ x \notin L \text{ is accepted} \end{array}\right\}$ with proba $\leq 1/3$.

This criterion defines the complexity class

BPP ("Banded-error probabilistic polynomial time")

= class of pbs solved by polynomial-size (uniform) randomized circuits.

↳ This is the analogous of P for randomized circuits.

Notes:

* it is clear that $P \subseteq BPP$ (a deterministic circuit is a special case of a randomized circuit in which the randomness is never consulted) (case $\delta = 1/2$)

* it is widely believed (but not proven) that $BPP = P$ which means that randomness does not expand the class of pbs that one can solve efficiently (with poly-size resources).

There is also a randomized version of the NP class called

MA ("Merlin - Arthur")

= class of pbs that can be easily checked by a randomized verifier when a suitable witness is provided.

rigorous definition:

A language $L \in MA$ iff $\exists$ poly-size randomized verifier $V(x,y)$ such that

if $x \in L \rightarrow \exists\, y$ with $\text{prob}(V(x,y)=1) \geq 2/3$

if $x \notin L \rightarrow \forall\, y$ $\text{proba}(V(x,y)=1) \leq 1/3$

Note:

* $BPP \subseteq MA$ (like $P \subset NP$) (no need for a witness)
* $NP \subseteq MA$ (special case with no randomness)

* Reversible classical computations

We saw that in classical computation many gates are irreversible, in particular the gates that take 2 bits of inputs to 1 bit of output

for ex : AND

↳ 00, 01, 10 → 0
11 → 1

OR

↳ 00 → 0
01, 10, 11 → 1

XOR : 00 → 0
01, 10 → 1
11 → 0

(or "addition modulo 2" $x_1 \oplus x_0$.)

These gates are <u>irreversible</u> because one cannot infer what the input was just from the knowledge of the output.
In other words, some of the information that was input to the gate has been lost when the gate operates

=> information has been <u>erased</u>.

On the contrary, in a reversible computation no info is erased, and one can always recover the input from the knowledge of the output.

In fact it turns out that any (Boolean) computation can be done using only reversible operations

i.e. a reversible computer can simulate the circuit model, and this can be done efficiently.

Note: Erasure has an energy cost. The Landauer principle from thermodynamics tells us that erasing a bit of information requires work

$$W \geq k_B T \ln 2$$

(k_B: Boltzmann constant; T: temperature of the environment)

But the energy cost associated with erasing bits can in fact be eliminated.

How ?

A reversible computer evaluates invertible fcts taking n bits to n bits :

$$f : \{0,1\}^n \longrightarrow \{0,1\}^n$$

f is invertible means that it has a unique input for each output and we can run the computation backward to recover the input from the output.

Note :

The set of possible inputs =
set of possible outputs =
all the bit strings of length n (2^n)

$\Rightarrow$ Can regard the action of an invertible fct as doing a permutation of the 2^n n-bit strings.

# of such invertible fcts
= # of permutations
$= (2^n)!$
$\approx \left(\frac{2^n}{e}\right)^{2^n}$ (Stirling approx) $\left(m! \approx \left(\frac{m}{e}\right)^m\right)$

(actually a small fraction of the total # of fcts, taking n bits to n bits

not necessarily invertible

$$\left(2^{2^n}\right)^n = \left(2^n\right)^{2^n}$$)

e^{-2^n} decreases very fast!

Now one can always encode a Boolean fct (which is non invertible) into an invertible one.
For instance say

$\tilde{f}$ is a Boolean fct : $x \longmapsto \tilde{f}(x)$

x: n-bit string. $\tilde{f}(x)$: 1-bit

it can be encoded into invertible f :

$$f(x,y) = \left(x,\ y \oplus \tilde{f}(x)\right)$$

$f(x,y)$: x n bits, y 1 bit; x n bits; $y \oplus \tilde{f}(x)$: 1 bit

where $\oplus$ denotes addition modulo 2.

$$\begin{cases} 0 \oplus 0 = 0 \\ 0 \oplus 1 = 1 \oplus 0 = 1 \\ 1 \oplus 1 = 0. \end{cases}$$

see that $\oplus$ is in fact the XOR gate

=> - the first argument x of the input is the same in the output

- the second argument y is
  - unchanged if $\tilde{f}(x) = 0$
  - flipped if $\tilde{f}(x) = 1$.

=> if we set $y = 0$ in the input → we get $f(x)$ in the output.

Note that f is indeed invertible because applying f twice undoes the bit flip:

$$f(f(x,y)) = f(x, y \oplus \tilde{f}(x))$$

$$= (x, y \oplus \tilde{f}(x) \oplus \tilde{f}(x))$$

$$= (x, y).$$

always 0 modulo 2

$$\Rightarrow f^2 = \mathbb{1} \quad \Rightarrow f = f^{-1}$$

f is actually self-inverse

Just like for Boolean fcts,
One can also ask if any reversible computation can be executed by circuits built from simple components -

=> Are there universal sets of reversible gates?

The answer is yes, but 1-bit → 1bit and 2-bit → 2-bit gates will not suffice -
We need 3-bit → 3-bit gates for classical universality -

Note :

This can be seen easily :

Out of the $(2^4)^2 = 256$ possible 2-bit → 2bit gates, $4! = 24$ are reversible (ex: CNOT) and these are all linear :

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \longmapsto \begin{pmatrix} x_0' \\ x_1' \end{pmatrix} = M \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix}$$

M is one of the 6 invertible 2x2 matrices with binary entries :

$$\begin{pmatrix}1&0\\0&1\end{pmatrix}, \begin{pmatrix}0&1\\1&0\end{pmatrix}, \begin{pmatrix}1&1\\0&1\end{pmatrix}, \begin{pmatrix}1&0\\1&1\end{pmatrix}$$

$$\begin{pmatrix}0&1\\1&1\end{pmatrix}, \begin{pmatrix}1&1\\1&0\end{pmatrix}$$

(these can be inverted by another matrix with binary entries).

=> These 24 linear operations account for the 24 invertible 2 bit → 2 bit gates.

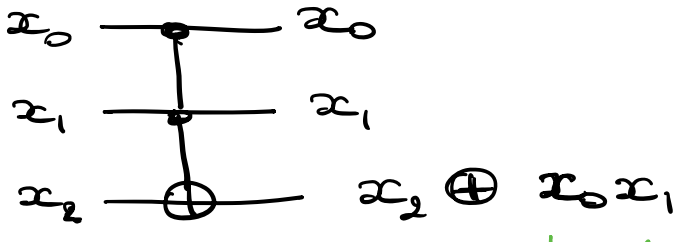
However we also need <u>non-linear</u> operations to achieve multiplication and for instance the AND gate.

=> need 3-bit → 3-bit reversible gate

for instance: The <u>Toffoli</u> gate

(here this is the classical version)

$(x_i = 0, 1)$



If we fix the input bits appropriately we can simulate

NOT $(x_0 = x_1 = 1)$

AND $(x_2 = 0)$

and OR can be constructed from NOT & AND

$\Rightarrow$ Toffoli is universal for Boolean logic

(and also universal for reversible computation)

It can be shown that the circuit for Boolean fct $f: \{0,1\}^n \to \{0,1\}$ with G irreversible gates can be simulated by reversible circuit of size $O(G + n)$

("Garbage removal Lemma")

$=$

$\downarrow$

only adds a linear scaling.

$\Rightarrow$ Boolean circuits can be simulated "efficiently" by reversible circuits

=> complexity classes P, NP ...
do not depend on whether we
choose a reversible or irreversible
model of classical comput°.

also true for PSPACE.

=> This helps us understand why
quantum circuits, which are always
reversible, can simulate any classical
computation with at least the same efficiency

and thus why quant. computation is
indeed a generalization of classical
computation.