Computability and Complexity

Richard Baron

Year 2017-2018

General notion : A Problem

Definition 1

(Decision problem) : A decision problem is defined by a set of instances I, and a subset $P \subseteq I$ called positive.

Intuitively, positive instances are "yes" answer to a given question

Example 1

Prime numbers : $I = \mathbb{N}$, $P = \{n \in \mathbb{N} | n \text{ is prime}\}$
Connected graphs : $I = \{G = (V, E) | G \text{ is a finite graph}\}$,
$P = \{G = (V, E) | G \text{ is connected}\}$
Automata acceptance : $I = \{(A, w) | A \text{ automaton}, w \text{ word}\}$,
$P = \{(A, w) | A \text{ accepts } w\}$

Numerous Real Problems are Optimization Problems

OPTIMIZATION PROBLEM
DATA : set X, objective function $f : X \mapsto \mathbb{R}$
QUESTION : maximize/minimize f on X $(\max_{x \in X} f(x)$ or $\min_{x \in X} f(x))$

(Associated) DECISION PROBLEM
DATA : set X, objective function $f : X \mapsto \mathbb{R}$, $K \in \mathbb{R}$
QUESTION : Is there $x \in X$ such that $f(x) \geq K$? (or $f(x) \leq K$ for the minimization prob.)

OPTIMIZATION is at least as difficult as DECISION

Proof:

Assume not: DECISION more difficult than OPTIMIZATION
e.g. algorithme A_{DEC} to solve DECISION in $O(n^5)$
and algorithm A_{OPT} to solve OPTIMIZATION in $O(n^3)$

then: apply A_{OPT} and compare the result with K ($O(n^3)$)
In other word:

solve OPTIMIZATION $\Longrightarrow$ answer to DECISION

Converse: NO (e.g. TSP)

From Problems to Languages

Definition 2

Let Σ be an alphabet. A encoding (or encoding scheme) is a one-to-one function from I to Σ^*. The encoding of $x \in I$ is denoted $\langle x \rangle$. The language associated with P is $L_P = \{\langle x \rangle | x \in P\}$.

Example 2

Numbers : $\langle n \rangle$ = decimal or binary encoding of n.
Graphs (...)

Turing machine

Two elements : (a finite control + a infinite tape) linked by a read/write head
Tape divided into squares, or cells.
Cell's elements indexed by $\mathbb{N}$

Definition 3

A Turing machine is defined by the sextuplet $(Q, \Sigma, \Gamma, E, q_0, F, \#)$*:*

- *a non empty set of states* Q*;*
- *an input alphabet* Σ *with* $\# \notin \Sigma$ *;*
- *a tape alphabet* Γ *containing all the symbols that can be written on the tape. Of course,* $\Sigma \subseteq \Gamma$ *and* $\# \in \Gamma$*;*
- *an set of transitions* E *of the form* (p, a, q, b, x)*, with* $p \in Q, q \in Q, a \in \Gamma, b \in \Gamma, x \in \{\lhd, \rhd\}$*. The transition may be represented by* $p, a \rightarrow q, b, x$*;*
- *an initial state* p_0*,* $p_0 \in Q$*;*
- *a set of final or accepting states* $F \subset Q$*;*
- *a blank symbol* $\#$ *that appears in all but a finite number of cells of the tape, those that hold input symbols.*

Turing machine

$\triangleleft$: move of the read/write head to the left (one cell)

$\triangleright$...

Transitions sometimes denoted by $p, a \to q, b, x$

Deterministic Turing Machine : at most **one** transition for each couple (p, a)

Non-deterministic Turing Machine : **several** possible transitions

Example 3

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, \#\}, \delta, q_0, \{q_4\}, \#)$
with δ defined by: 7

	Symbol				
State	*0*	*1*	*X*	*Y*	*#*
q_0	$q_1, X, \triangleright$	—	—	$q_3, Y, \triangleright$	—
q_1	$q_1, 0, \triangleright$	$q_2, Y, \triangleleft$	—	$q_1, Y, \triangleright$	—
q_2	$q_2, 0, \triangleleft$	—	$q_0, X, \triangleright$	$q_2, Y, \triangleleft$	—
q_3	—	—	—	$q_3, Y, \triangleright$	$q_4, \#, \triangleright$
q_4	—	—	—	—	—

A configuration defined by :

1. state of the machine (element of Q);
2. symbols of the tape;
3. position of the read/write head.

Configuration uqv : Machine in state q, u word on the tape (before head), v word beginning at the position of the head.
One step (move) of a computation :

1. change the state;
2. write a new symbol on the tape;
3. move the read/write head of one position.

Definition 4

A step of a computation is a pair (C, C') denoted $C \rightarrow C'$ such that :

- *either $C = ucpav$ and$C' = uqcbv$ and the transition is $p, a \rightarrow q, b, \lhd$*
- *either $C = upav$ and$C' = ubqv$ and the transition is $p, a \rightarrow q, b, \rhd$*

Definition 5

A computation is a serie of successives configurations $C_0 \rightarrow C_1 \rightarrow \ldots \rightarrow C_k$. A computation is accepting if C_0 is an initial configuration, that is $C_0 = q_0w$ with $w \in \Sigma^*$ and C_k is final, that is $C_k = uqv$ with $q \in F$.

Turing Machine :

- Accepting. input = word, output = Yes/No
- Computing. input = word, output : word(s)

Definition 6

$w \in \Sigma^*$ is accepted by a Turing machine $\mathcal{M}$ if there is an accepting computation with initial configuration $q_0 w$. The set of accepted words of $\mathcal{M}$ is denoted $L(\mathcal{M})$.

Note: Alternatively, F may be partitioned into F_Y and F_N

Definition 7

A two-way infinite tape machine is formally identical to the preceding model but with cell's tape indexed by $\mathbb{Z}$ (not $\mathbb{N}$)

Proposition 2.1

A two-way infinite tape machine is equivalent to a one-way infinite tape machine. Reciprocally, A one-way infinite tape machine is equivalent to a two-way infinite tape machine.

Proof:

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
$	b	a	X	#	#	#	...

and:

- $\Gamma' = \Gamma \times (\Gamma \cup \{\$\})$ (head read both symbols of new cells)
- $\Sigma' = \Sigma \times \{\$, \#\}$ (input is encoded on **upper part** of cells)
- $\#' = (\#, \#)$
- $Q' = Q \times \{\uparrow, \downarrow\}$
- $q_0' = (q_0, \uparrow)$
- $F' = F \times \{\uparrow, \downarrow\}$

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
$	b	a	X	#	#	#	...

New transitions in E' ?

Assume $(p, a, q, b, \rhd) \ \in E$

Then we create:

$((p, \uparrow), (a, .), (q, \uparrow), (b, .), \rhd)$ **and** $((p, \downarrow), (., a), (q, \downarrow), (., b), \lhd)$ in E'

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
\$	b	a	X	#	#	#	...

New transitions in E' ?
Assume $(p, a, q, b, \lhd) \ \in E$
Then we create (if $. \neq \$$ (see Special cases below)):
$((p, \uparrow), (a, .), (q, \uparrow), (b, .), \lhd)$ **and** $((p, \downarrow), (., a), (q, \downarrow), (., b), \rhd)$
in E'

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
$	b	a	X	#	#	#	...

Problem: Special cases for cell 0

- Assume $(p, a, q, b, \lhd) \in E$ and head of M' is positioned onto this cell
  Create $((p, \uparrow), (a, \$), (q, \downarrow), (b, \$), \rhd) \in E'$
- Plus $((p, \downarrow), (., \$), (p, \uparrow), (., \$), \triangledown) \in E'$ $(\triangledown = \rhd + \lhd)$

Definition 8

A multitape machine has k tapes, with k corresponding read/write head. A transition is an element of the set
$Q \times \Gamma^k \times Q \times \Gamma^k \times \{\triangleright, \triangleleft, \triangledown\}^k$

Proposition 2.2

Every multitape Turing machine $\mathcal{M}$ is equivalent to one-tape Turing machine $\mathcal{M}'$ that accepts the same inputs.

solution 1:

- M' stores contents of k tapes separated by new symbol $\$ \notin \Gamma$
- to keep track of heads locations, M' inserts a symbol $\downarrow$ before every symbols located under one head
- M' scans the whole content
- M' applies the required transition of state
- M' makes second pass to update (k) contents

solution 2: instead of $\downarrow$, new "dotted" symbols are added to Γ and indicate location of one head

solution 3: k tapes are replaced by one, where each cell contains a symbol of Γ^k (same idea as the bi-infinite machine). Additionally, k symbols $0/1$ are added to keep track of heads locations

Turing machine

				↓			
Tape 1 0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...

		↓					
Tape 2 0	1	2	3	4	5	6	
X	a	X	b	#	#	#	...

Simulated by

	↓						
0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
0	0	0	0	1	0	0	
X	a	X	b	#	#	#	...
0	0	1	0	0	0	0	

(M' reads symbol A0a0)

TURING MACHINE

Posted on July 2, 2014 by bytesoftheday

Question #12: Following is a transition table of non-deterministic TM:

δ	0	1	B
q_0	$\{(q_0, 1, R)\}$	$\{(q_1, 0, R)\}$	$\emptyset$
q_1	$\{(q_1, 0, R),\ (q_0, 0, L)\}$	$\{(q_1, 1, R),\ (q_0, 1, L)\}$	$\{(q_2, B, R)\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$

Which all of the ID's can be reached after third transitions, when the initial ID is q_0011?

Options:

1. $100q_1$, $1q_001$
2. $11q_01$
3. $101q_1$, $1q_001$
4. $111q_1$

Proposition 2.3

For every Turing machine $\mathcal{M}$ there exists a Turing machine $\mathcal{M}'$ such that :

1. $L(\mathcal{M})=L(\mathcal{M}')$ *and* $\mathcal{M}$ *halts if and only if* $\mathcal{M}'$ *halts ;*
2. $\mathcal{M}'$ *has two states* q_+ *and* q_- *such that :*
   - $F' = \{q_+\}$
   - $\mathcal{M}'$ *always halts in* q_+ *and* q_-
   - $\mathcal{M}'$ *halts only in either* q_+ *or* q_-

Definition 9

A Non-deterministic Turing machine is defined by the sextuplet $(Q, \Sigma, \Gamma, \Delta, q_0, F, \#)$*:*

- *a non empty set of states* Q*;*
- *an input alphabet* Σ *with* $\# \notin \Sigma$ *;*
- *a tape alphabet* Γ *containing all the symbols that can be written on the tape. Of course,* $\Sigma \subseteq \Gamma$ *and* $\# \in \Gamma$*;*
- *an transition relation* $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\lhd, \rhd\}$
- *an initial state* p_0*,* $p_0 \in Q$*;*
- *a set of final or accepting states* $F \subset Q$*;*
- *a blank symbol* $\#$ *that appears in all but a finite number of cells of the tape, those that hold input symbols.*

Non-deterministic : accepting if **at least** one accepting computation

Proposition 2.4

Every non-deterministic Turing machine $\mathcal{M}$ is equivalent to a deterministic Turing machine $\mathcal{M}'$. If $\mathcal{M}$ has no infinite computation, then $\mathcal{M}'$ has no infinite computation.

Proof:

Idea: $\mathcal{M}'$ will simulate all possible computations of $\mathcal{M}$
BUT must not get trapped into infinite ones (if any)
$\mathcal{M}'$ has 3 tapes:

1. store input word x
2. encode the choices made during simulation
3. working tape

Define $r = \max_{q \in Q, a \in \Sigma} | \{(q, a, q', z, Z) \in \Delta\}|$
r is the greatest number of possible transitions of the NDTM $\mathcal{M}$, for all combinations of state q and symbol a on the tape (for all cells of transition table)

Step t of DTM $\mathcal{M}$' :

- erase content of 3rd tape, then copy input word x on this tape
- generate the string $y \in \{1, \ldots, r\}^*$ numbered t (in lexicographic order) $y = m_{i_1} m_{i_2} \ldots m_{i_l}$ and write it on 2nd tape
- simulate the computation of NDTM $\mathcal{M}$ for at most l steps (recall $l = |y|$). At step j ($1 \leq j \leq l$) of this simulation, use m_{i_j} to select which transition (of NDTM $\mathcal{M}$) is to be applied. If less than m_{i_j} possible transitions, then skip to $t + 1$
- if simulation of NDTM $\mathcal{M}$ is such that $\mathcal{M}$ reaches an accepting state, then DTM $\mathcal{M}'$ accepts x. Else, $\mathcal{M}'$ skips to step $t + 1$.

Comment: $\{1, \ldots, r\}^*$ is the (infinite) set of all strings written with symbols of $\{1, \ldots, r\}$.
Lexicographic ordering of $\{1, \ldots, 4\}^*$ is

$$\begin{aligned} &\{1, 2, 3, 4, 11, 12, 13, 14, 21, 22, 23, 24, 31, 32, \ldots \\ &\ldots, 43, 44, 111, 112, 113, 114, 121, 122, \ldots\} \end{aligned} \tag{1}$$

Number of operations of this simulation ?
Assume executions $\mathcal{M}$ is finite, bounded by $\bar{l}$ (number of steps)
Time for simulation by $\mathcal{M}'$?
A string $y = m_{i_1} m_{i_2} \dots m_{i_l}$ (thus $|y| = l$) encodes an execution (of $\mathcal{M}$) in **at most** l steps
Thus, total time for this simulation by $\mathcal{M}'$ is bounded by
$T = \sum_{l=1}^{\bar{l}} l r^l$
Rewrite $T = r \sum_{l=1}^{\bar{l}} l r^{l-1}$
Let $f_n(r) = \sum_{l=1}^{n} l r^{l-1}$.

Turing machine

$$\begin{aligned}
(1-r)f_n(r) &= \sum_{l=1}^{n} l r^{l-1} - \sum_{l=1}^{n} l r^{l} \\
&= \sum_{j=0}^{n-1}(j+1)r^j - \sum_{j=1}^{n} j r^j \\
&= r^0 + \sum_{j=1}^{n-1}\left((j+1)r^j - j r^j\right) - n r^n \\
&= r^0 + \sum_{j=1}^{n-1} r^j - n r^n \\
&= \sum_{j=0}^{n-1} r^j - n r^n \\
&= r^0 \frac{1-r^n}{1-r} - n r^n
\end{aligned}$$

Then

$$f_n(r) = \frac{1 - r^n}{(1 - r)^2} - n\frac{r^n}{1 - r}$$

and

$$T = r\left[\frac{1 - r^{\bar{l}}}{(1 - r)^2} - \bar{l}\frac{r^{\bar{l}}}{1 - r}\right]$$

That is $T = O(r^{\bar{l}})$
Recalling:

$$\begin{aligned} r^\alpha &= \exp(\alpha \log(r)) \\ &= \exp(\alpha \log_2(r)\log(2)) \\ &= 2^{\alpha \log_2(r)} \end{aligned}$$

we have

$$T = O(r^{\bar{l}}) = O(2^{\bar{l}\log_2(r)}) = O(2^{\bar{l}})$$

Definition 10
A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a Turing machine $\mathcal{M}$ such that $L = L(\mathcal{M})$. Equivalently, a problem P is recursively enumerable if L_P is recursively enumerable.

Definition 11
An enumerator is a deterministic Turing machine that writes some words of Σ^* on an output tape. These words are separated by a symbol $\$ \notin \Sigma$.

Proposition 3.1

A language $L \subseteq \Sigma^*$ is recursively enumerable if and only if L is the set of words enumerated by an enumerator.

Proposition 3.2

If languages L and L' are recursively enumerable, then languages $L \cup L'$ and $L \cap L'$ are recursively enumerable.

Definition 12

A language $L \subseteq \Sigma^*$ is decidable if there exists a Turing machine $\mathcal{M}$ without infinite computation and such that $L = L(\mathcal{M})$. $\mathcal{M}$ decides L.

Finite computation !

Proposition 4.1

If two languages $L, L' \in \Sigma^*$ are decidable, then $L \cup L'$, $L \cap L'$ and $\Sigma^* \setminus L$ are decidable.

Proposition 4.2

Let $L \subseteq \Sigma^*$. If L and its complement set $\Sigma^* \setminus L$ are recursively enumerable, then they are decidable.

$\mathcal{M}$ a machine, w a word
Recall that $\langle x \rangle$ denotes an encoding of x

Definition 13

Let $L_{\in} = \{\langle M, w \rangle | w \in L(\mathcal{M})\}$. $L_{\in}$ is recursively enumerable. A Turing machine $\mathcal{M}_U$ such that $L(\mathcal{M}_U) = L_{\in}$ is called universal.

Proposition 4.3

$L_{\in}$ is not decidable.

Corollaire 1

$\overline{L}_{\in}$ is not recursively enumerable.

Definition 14

A function $f : \Sigma^* \to \Gamma^*$ is computable if there exists a machine that for each word $w \in \Sigma^*$ halts with the word $f(w) \in \Gamma^*$ written on the tape.

Definition 15

Let A et B two problems with alphabets Σ_A and Σ_B respectively and languages L_A and L_B respectively. A reduction from A to B is a function $f : \Sigma_A^* \to \Sigma_B^*$ computable and such that $w \in L_A \Leftrightarrow f(w) \in L_B$. We denote $A \leq_m B$.

Proposition 4.4

If $A \leq_m B$ and if B is decidable, then A is decidable.

Corollaire 2

If $A \leq_m B$ and if A is undecidable, then B is undecidable.

Definition 16 (Complexity)

Let $\gamma = q_0 w \to C_0 \to C_1 \to \ldots \to C_m$ be a computation of a Turing machine $\mathcal{M}$ on an input word w.

1. *the time $t_{\mathcal{M}}(\gamma)$ of this computation is m*
2. *the space $s_{\mathcal{M}}(\gamma)$ is the number of cells visited by the head during the computation*

Complexity for an input w:

$$t_{\mathcal{M}}(w) = \max_{\gamma} t_{\mathcal{M}}(\gamma) \quad \text{et} \quad s_{\mathcal{M}}(w) = \max_{\gamma} s_{\mathcal{M}}(\gamma)$$

Complexity (at worst case):

$$t_{\mathcal{M}}(n) = \max_{|w|=n} t_{\mathcal{M}}(w) \quad \text{et} \quad s_{\mathcal{M}}(n) = \max_{|w|=n} s_{\mathcal{M}}(w)$$

A first bound

Lemma 1

For every one tape Turing machine $\mathcal{M}$, there exists a constant K s.t.

$$s_{\mathcal{M}}(n) \leq t_{\mathcal{M}}(n) \;\; et \;\; t_{\mathcal{M}}(n) \leq 2^{Ks_{\mathcal{M}}(n)}$$

Proof: For a computation with input word w, any configuration C_i composed of symbols of Γ and Q (only one state), that is
$C_i \in (\Gamma \cup Q)^*$
We assume here that $s_{\mathcal{M}}(w) > |w|$
We have $|C_i| \leq s_{\mathcal{M}}(w) + 1$ thus
$|\{C_i, 0 \leq i \leq t_{\mathcal{M}}(w)\}| \leq |\Gamma \cup Q|^{s_{\mathcal{M}}(w)+1} \leq |\Gamma \cup Q|^{k.s_{\mathcal{M}}(w)}$
with k a large enough integer
Note that all configurations are different from each other
then $|\{C_i, 0 \leq i \leq t_{\mathcal{M}}(w)\}| = t_{\mathcal{M}}(w) + 1$
Thus $t_{\mathcal{M}}(w) \leq t_{\mathcal{M}}(w) + 1 \leq |\Gamma \cup Q|^{k.s_{\mathcal{M}}(w)} \leq 2^{Ks_{\mathcal{M}}(w)}$
for a constant K

Proof: If $s_{\mathcal{M}}(w) \leq |w|$ then $|C_i| \leq |w| + 1$
We can choose k large enough such that $k.s_{\mathcal{M}}(w) \geq |w|$
We have $|\{C_i, 0 \leq i \leq t_{\mathcal{M}}(w)\}| \leq |\Gamma \cup Q|^{|w|+1}$
Note that all configurations are different from each other
then $|\{C_i, 0 \leq i \leq t_{\mathcal{M}}(w)\}| = t_{\mathcal{M}}(w) + 1$
Thus
$t_{\mathcal{M}}(w) \leq t_{\mathcal{M}}(w) + 1 \leq |\Gamma \cup Q|^{|w|+1} \leq |\Gamma \cup Q|^{k.s_{\mathcal{M}}(w)} \leq 2^{Ks_{\mathcal{M}}(w)}$
for K large enough

Bi-infinite tape TM: one transition of the bi-infinite tape TM = one transition for the simulating TM

Multi-tape TM:

Proposition 5.1

Every k-tape Turing machine $\mathcal{M}$ is equivalent to a one-tape Turing machine $\mathcal{M}'$ s.t.

$$t_{\mathcal{M}'}(n) = \mathcal{O}(t^2_{\mathcal{M}}(n))$$

Non-deterministic TM: r maximal cardinality of the sets $\delta(p, a) = \{(q, b, x) | p, a \to q, b, x \in E\}, \forall p \in Q, a \in \Gamma$

Proposition 5.2

Every (non-deterministic) Turing machine $\mathcal{M}$ is equivalent to a determistic Turing machine $\mathcal{M}'$ s.t.

$$t_{\mathcal{M}'}(n) = 2^{\mathcal{O}(t_{\mathcal{M}}(n))}$$

Definition 17

Given $f : \mathbb{N} \to \mathbb{R}^+$, we define :

- *$TIME(f(n))$ the set of problems decided by a deterministic Turing machine in time $\mathcal{O}(f(n))$*
- *$NTIME(f(n))$ the set of problems decided by a non-deterministic Turing machine in time $\mathcal{O}(f(n))$*

Definition of classes :

$$P = \bigcup_{k \geq 0} TIME(n^k)$$

$$NP = \bigcup_{k \geq 0} NTIME(n^k)$$

Proposition 5.3

$$P \subseteq NP$$

Definition 18

A polynomial-time verifier for a language L is a deterministic Turing machine $\mathcal{M}$ which accepts input $\langle w, c\rangle$ in polynomial time in $|w|$ such that $L = \{w | \exists c, \langle w, c\rangle \in L(\mathcal{M})\}$

Polynomial time in $|w| \Rightarrow$ size of c is polynomial

$c =$ certificate/witness

Algorithmically more intuitive

Proposition 5.4

Language L is in NP if and only if there exists a polynomial-time verifier for L.

Alternative view

Proof:

First, assume $L \in NP$. There is non-deterministic machine $\mathcal{M}$ which accepts L in polynomial time.

Then verifier $\mathcal{V}$ has input w, and c can be an encoding of the transitions of $\mathcal{M}$ when $\mathcal{M}$ accepts w.

$\mathcal{V}$ accepts w if $\mathcal{M}$ accepts w

Second, assume there is a verifier $\mathcal{V}$ for L

Given w, a machine $\mathcal{M}$:

- picks a random c which size is polynomially bounded in $|w|$
- simulates $\mathcal{V}$ with input $< w, c >$

$\mathcal{M}$ accepts w if $\mathcal{V}$ accepts $< w, c >$

As $\mathcal{V}$ computes in polynomial time, $\mathcal{M}$ computes in polynomial time

Question : $NP \subseteq P$.
If so : $P = NP$
1 M$ question, see Millennium problem at Clay Mathematics Institute
Prove that $\exists L \in NP$ with $L \notin P$? Not until today.
Instead : L NP-complete ?

Definition 19 (Polynomial reduction)

Let A and B be problems, with respective languages L_A and L_B on alphabets Σ_A and Σ_B. A polynomial reduction (transformation) from A to B is a polynomial-time computable function $f : \Sigma_A^* \to \Sigma_B^*$ such that :

$$w \in L_A \Longleftrightarrow f(w) \in L_B;$$

This is denoted $A \leq_P B$.

$\leq_P$: transitive and reflexive

$\leq_P$ is transitive

Proof:

assume $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then

$$\exists \text{ a DTM } \mathcal{M}, \forall x \in \Sigma_1^*, \mathcal{M}(x) \in L_2 \iff x \in L_1$$

with polynom p, s.t. $t_{\mathcal{M}}(x) \leq p(|x|)$.

$$\exists \text{ a DTM } \mathcal{M}', \forall x \in \Sigma_2^*, \mathcal{M}'(x) \in L_3 \iff x \in L_2$$

with polynom p', s.t. $t_{\mathcal{M}'}(x) \leq p'(|x|)$

Let $x \in \Sigma_1^*$,

$$\mathcal{M}'(\mathcal{M}(x)) \in L_3 \iff \mathcal{M}(x) \in L_2 \iff x \in L_1$$

and $t_{\mathcal{M}'}(\mathcal{M}(x)) \leq p'(|\mathcal{M}(x)|) \leq p'(p(|x|)) = p''(|x|)$

using $|\mathcal{M}(x)| \leq t_{\mathcal{M}}(x) \leq p(|x|)$ and p' is increasing

Proposition 6.1

If $A \leq_P B$ and $B \in P$, then $A \in P$.

Proof:

if $A \leq_P B$ then:

$$\exists \text{ a DTM } \mathcal{M}, \forall x \in \Sigma_1^*, \mathcal{M}(x) \in L_B \iff x \in L_A$$

and polynom p, s.t. $t_{\mathcal{M}}(x) \leq p(|x|)$

Furthermore, if $B \in P$ then

$$\exists \text{ a DTM } \mathcal{M}', \forall x \in \Sigma_2^*, \mathcal{M}'(x) = \text{YES} \iff x \in L_B$$

and polynom p', s.t. $t_{\mathcal{M}'}(x) \leq p'(|x|)$.

Let $x \in L_A$. Then $\mathcal{M}(x) \in L_B$. Converse is also true, and

$$|\mathcal{M}(x)| \leq t_{\mathcal{M}}(x) \leq p(|x|) \tag{2}$$

$\mathcal{M}'(\mathcal{M}(x)) = \text{YES} \iff \mathcal{M}(x) \in L_B \iff x \in L_A$. and

$t_{\mathcal{M}'}(\mathcal{M}(x)) \leq p'(|\mathcal{M}(x)|) \leq p'(p(|x|)) = p''(|x|)$.

Definition 20

A problem A is NP-hard if for every problem $B \in NP$, $B \leq_P A$ (every B is reducible to A). If additionally $A \in NP$ then A is NP-complete.

Proposition 6.2

If A is NP-hard and $A \leq_P B$ then B is NP-hard

Proof:

If A is NP-hard then $\forall L' \in NP,\ L' \leq_P A$

By transitivity of $\leq_P$, $A \leq_P B$ implies $\forall L' \in NP,\ L' \leq_P B$

Example

HC $\leq_P$ TSP
Proof :
Standard formulation for Decision Problems

TRAVELING SALESMAN PROBLEM (TSP)
INSTANCE : a finite set $C = \{c_1, c_2, \ldots, c_m\}$ of cities, distances $d(c_i, c_j) \in \mathbb{N}$ for every $(c_i, c_j) \in C \times C$, a bound $B \in \mathbb{N}$.
QUESTION : is there a tour of all the cities of C having a length of at most B, that is, a permutation π de $\{1, \ldots, m\}$ s.t.

$$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B$$

EXAMPLE

HAMILTONIAN CIRCUIT (HC)
INSTANCE : a graph $G = (V, E)$ with $|V| = n$ (vertices are denoted v_i)
QUESTION : is there an hamiltonian circuit in G, that is, a permutation π of $\{1, \ldots, n\}$ s.t.
$\forall i \in \{1, \ldots, n-1\}, (v_{\pi(i)}, v_{\pi(i+1)}) \in E$ and $(v_{\pi(n)}, v_{\pi(1)}) \in E$

Methodology: **for any instance** of **HC** define ONE peculiar instance of **TSP**

Set $C = V$
and $\forall v_i, v_j \in V \times V$,

$$d(v_i, v_j) = \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 2 & \text{sinon} \end{cases}$$

Finally $B = |V|$.

Example

Step 1: Check that the construction is polynomial.
Here: $O(|V|^2)$
Step 2: Recall

Definition 21 (Polynomial reduction)

Let A and B be problems, with respective languages L_A and L_B on alphabets Σ_A and Σ_B. A polynomial reduction (transformation) from A to B is a polynomial-time computable function $f : \Sigma_A^* \to \Sigma_B^*$ such that :

$$\mathbf{w} \in \mathbf{L_A} \iff \mathbf{f(w)} \in \mathbf{L_B}$$

This is denoted $A \leq_P B$.

Example

Step 2.1:
Assume the instance of HC is positive
That is, there is a permutation π of the vertices of G which defines an hamiltonian circuit.
Let us take the same permutation π on THE **TSP** instance.
Circuit = only existing edges of G
Distance is $|V| = B$.
TSP instance is positive

EXAMPLE

Step 2.2:
Assume the instance of TSP is positive
The there is a permutation π de $\{1, \ldots, |V|\}$ such that

$$\sum_{i=1}^{|V|-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(|V|)}, c_{\pi(1)}) \leq B = |V|$$

Prove it uses only distances of 1.
We denote the preceding equation: $\sum_{j=1}^{|V|} d_j \leq |V|$
Assume there is a distance different from 1
Thus, it exists $j_0 \in \{1, \ldots, |V|\}$ s.t. $d_{j_0} = 2$ and
$\sum_{j=1}^{|V|} d_j = d_{j_0} + \sum_{j=1, j \neq j_0}^{|V|} d_j \leq |V|$, thus $\sum_{j=1, j \neq j_0}^{|V|} d_j \leq |V| - 2$
But $\forall j, j \neq j_0$ $d_j \geq 1$ implies $\sum_{j=1, j \neq j_0}^{|V|} d_j \geq |V| - 1$
A contradiction
Thus, the tour uses only distances of 1, which correspond to existing edges of G
This defines a hamiltonian circuit in G
HC instance is positive

Example

Steps 1, 2.1, 2.2 $\implies$ polynomial transformation from **HC** to **TSP**

TSP is NP-hard

Step 3: is **TSP** in NP ?
certificate ?
complexity of checking phase ?

TSP is NP-complete

Methodology

Assuming A is NP-complete
Proving NP-completeness of B by $A \leq_P B$

Step 1 Prove that the construction (of instance of B) is polynomial

Step 2 Prove

- **Step 2.1** Positive instance of $A \implies$ Positive instance of B
- **Step 2.2** Positive instance of $B \implies$ Positive instance of A

Now, as A is NP-hard, then B is NP-hard (Proposition 6.2)

Step 3 Prove B is in NP

CLIQUE
INSTANCE : a graph $G = (V, E)$ and an integer $J \leq |V|$
QUESTION : is there a subset $V' \subseteq V$ with $|V'| \geq J$ and such that $\forall u \in V', v \in V', (u, v) \in E$

From **3-SAT** (NP-complete)
3-SAT
INSTANCE : a set C of k clauses (denoted $C_j, j \in \{1, \ldots, k\}$) each of which is a disjunction of 3 literals (denoted C_{j1}, C_{j2}, C_{j3}), a set U of boolean variables denoted x_i. Each literal can be either a boolean variable x_i or its negation $\bar{x}_i$ (also written $\neg x_i$).
QUESTION : is there a truth assignment for U, that is a function $f : U \mapsto \{True, False\}$ such that the k clauses of C are True.

Redution:

CLIQUE is fully defined by $G = (V, E)$ **and** J

- definition of V: if literal C_{jl} is x_i (resp. $\neg x_i$), add a vertex labelled with the literal x_i (resp. $\neg x_i$) in V
- definition of E: add an edge from a vertex corresponding to literal C_{jl} to another vertex corresponding to literal C_{hg} if
  - $h \neq j$. No edge between any two vertices corresponding to literals from a same clause
  - literals C_{jl} and C_{hg} are not negations of one another (e.g. if C_{jl} is x_i and C_{hg} is $\neg x_i$ then no edge)
- definition of J: $J = k$

Step 1: reduction is polynomial
$|V| = 3k$ (done !)
Note : $|E| \leq 3k.3(k-1)/2$

Step 2: **3SAT** $\geq 0 \iff$ **CLIQUE** ≥ 0

Step 2.1: Assume instance of **3SAT** is positive
Then, there is a truth assignment which makes all clauses True
Thus, at least ONE literal per clause has a value True
V' = Pick one vertex per group of 3 corresponding to literals with value True
Let us show that V' is a CLIQUE of size J:

- clearly $|V'| = J$, as we picked exactly one literal per clause
- there is no edge between any two of them, as the corresponding literals can not be negation one from another (all of them has a value True)

Step 2.2: Assume THE instance (the one we built) of **CLIQUE** is positive

There is $V' \subseteq V$ with $|V'| \geq J$ s.t. $\forall u \in V', v \in V', (u, v) \in E$

Set corresponding literals to a value True

Remark: a CLIQUE (in this graph) has **at most** $J = k$ vertices, every vertex picked inside a group of 3 (corresponding to one clause) (proof by contradiction: assume $k + 1$ vertices, pigeon-hole principle...)

Thus $|V'| = J$

Let us show this defines a truth assignment s.t. all clauses are True

- can not assign value True to x_i and $\neg x_i$ at the same time: no edge between corresponding vertices $\Rightarrow$ truth assignment
- as **exactly** one vertex per group is in V', the corresponding literal being True, every clause is True

QED

From steps 1 and 2: **3SAT** $\leq_P$ **CLIQUE**
As **3SAT** is NP-complete (thus NP-hard), **CLIQUE** is NP-hard

Step 3: Show **CLIQUE** $\in$ **NP**
certificate $= V' \subseteq V$

- check $|V'| \geq J$: $O(|V|)$
- check $\forall u \in V', v \in V', (u, v) \in E$: $O(|V|^2)$

CLIQUE $\in$ **NP**

From steps 1, 2 and 3: **CLIQUE** is **NP-complete**

VERTEX COVER
INSTANCE : a graph $G = (V, E)$ and an integer $J \leq |V|$
QUESTION : is there a subset $V' \subseteq V$ with $|V'| \leq J$ and such that $\forall u \in V', v \in V', u \in V'$ or $v \in V'$

From...
3-SAT
INSTANCE : a set C of k clauses (denoted $C_j, j \in \{1, \ldots, k\}$) each of which is a disjunction of 3 literals (denoted C_{j1}, C_{j2}, C_{j3}), a set U of l boolean variables denoted x_i. Each literal can be either a boolean variable x_i or its negation $\neg x_i$ (sometimes written $\bar{x}_i$).
QUESTION : is there a truth assignment for U, that is a function $f : U \mapsto \{True, False\}$ such that the k clauses of C are True.

VERTEX COVER

Given an instance of **3-SAT** (C the set of clauses), instance of **VERTEX COVER** ($G = (V, E)$ and J) is defined by:

1. V is composed of:
   1.1 two vertices x_i et $\neg x_i$ for all $x_i \in U$;
   1.2 three vertices c_{j1}, c_{j2}, c_{j3} corresponding to C_{j1}, C_{j2}, C_{j3} for every clause C_j;
2. E is composed of:
   2.1 an edge $(x_i, \neg x_i)$ for every couple $x_i, \neg x_i$;
   2.2 three edges (c_{j1}, c_{j2}),(c_{j1}, c_{j3}),(c_{j2}, c_{j3}) for every triplet c_{j1}, c_{j2}, c_{j3} ;
   2.3 an edge between vertex c_{ji} and vertex x_r (resp. $\neg x_r$) if literal C_{ji} is x_r (resp. $\neg x_r$) ;
3. $J = l + 2k$ (recall $l = |U|$).

Example: assume $C = (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$

Step 1: reduction is polynomial
Recall $l = |U|$ and $k = |C|$
$|V| = 2l + 3k$ (done !)
Note : $|E| \leq (2l + 3k).(2l + 3k - 1)/2$
More precisely, here: $|E| = l + 3k + 3k$
Computation of J is also $O(l + k)$

Step 2.1: Assume the instance of **3-SAT** is positive

VERTEX COVER

There is a truth assignment such that the k clauses of C are True. We define a Vertex Cover V' of G by choosing the following vertices:

1. if variable x_i (resp. $\neg x_i$) is True, then choose vertex x_i (resp. $\neg x_i$) of G ;
2. among the three vertices c_{j1}, c_{j2}, c_{j3}, choose two of them, s.t. the one **which is not chosen** shares an edge with a vertex chosen at the first step.

Is it a Vertex Cover ?

1. every edge $(x_i, \neg x_i)$ has exactly one vertex in V' ;
2. every one of the edges (c_{j1}, c_{j2}),(c_{j1}, c_{j3}),(c_{j2}, c_{j3}) have at least one vertex in V' by the second step of the picking procedure
3. concerning edges between c_{ji} and x_r, two possibilities:
   3.1 $c_{ji} \in V'$;
   3.2 if not, then it shares an edge with a vertex $x_r \in V'$

Finally, note that we have picked exactly $l + 2k$ vertices in V' (i.e. J vertices)

Step 2.2: Assume the instance of **VERTEX COVER** is positive

VERTEX COVER

There is a set $V' \subseteq V$ with $|V'| \leq J = l + 2k$
First note that $l + 2k$ is the minimal size for a **VERTEX COVER** thus $|V'| = l + 2k$ (structure ?)
If vertex $x_i \in V'$ (resp. $\neg x_i \in V'$) set variable x_i (resp. $\neg x_i$) to value True
Now for every triangle (c_{j1}, c_{j2}, c_{j3}), consider THE vertex (say c_{jh}) which is not in V'
Then consider edge (c_{jh}, x_i) (resp. $(c_{jh}, \neg x_i)$). As $c_{jh} \notin V'$ we must have $x_i \in V'$ (resp. $\neg x_i \in V'$)
As variable x_i (resp. $\neg x_i$) has then a value True, clause C_j is True
Instance of **3-SAT**

Step 3: **VERTEX COVER** is in NP ?
certificate ? V' !
checking that $\forall (u, v) \in E$ we have $u \in V'$ or $v \in V'$ is $O(|V|^2)$

Theorem 1

SAT *is NP-complete*

Proof: we must show that $\forall A \in NP, A \leq_P$ **SAT**

Let w a positive instance of A. As $A \in NP$ there is a Non-Deterministic Turing Machine accepts w in polynomial time

Encode this computation into a propositional formula True if and only if w is positive

Denote $w = w_1 \dots w_n$

Let $M = (Q, \Sigma, \Gamma, E, q_0, F, \#)$ the Non-Deterministic Turing Machine

p a polynomial such that $t_M(n) \leq p(n)$.

An accepting computation of w is composed of at most $p(n) + 1$ configurations

Configuration = state + tape's content + head's position

Store information in:

1. table R of dimension $(p(n)+1) \times (p(n)+1)$ for symbols of the tape. $R(i,j)$ = symbol in cell j at step i
2. a vector Q of dimension $p(n)+1$. $Q(i)$ = state of M at step i
3. a vector P of dimension $p(n)+1$. $P(i)$ = position of the head at step i

Must be encoded in a propositional formula with boolean variables:

1. $r_{ij\alpha}$ with $0 \leq i,j \leq p(n)$ et $\alpha \in \Gamma$;
2. $q_{i\kappa}$ with $0 \leq i \leq p(n)$ et $\kappa \in Q$;
3. p_{ij} with $0 \leq i,j \leq p(n)$.

Number of variables: $O(p^2(n))$

1. $r_{ij\alpha}$ is True if $R(i,j)$ contains symbol α ;
2. $q_{i\kappa}$ is True if $Q(i)$ is κ ;
3. p_{ij} is True if $P(i)$ is j.

Computation = set of logical constraints on these variables, which must all be True

e.g. only one α such that $r_{ij\alpha}$ is True, only one κ such that $q_{i\kappa}$ is True...

$$\bigwedge_{0\le i,j\le p(n)} \left[\left(\bigvee_{\alpha\in\Sigma} r_{ij\alpha} \right) \wedge \bigwedge_{\alpha'\neq\alpha\in\Sigma} (\neg r_{ij\alpha} \vee \neg r_{ij\alpha'}) \right] \tag{3}$$

Example: Set $\Sigma = \{a, b, c\}$.

Formula is

$$(r_{ija} \vee r_{ijb} \vee r_{ijc}) \wedge ((\neg r_{ija} \vee \neg r_{ijb}) \wedge (\neg r_{ija} \vee \neg r_{ijc}) \wedge (\neg r_{ijb} \vee \neg r_{ijc}))$$

Verify that only $r_{ij\alpha}$ must be True