P, NP, and NP-completeness

16/1/2006

Running time (or *time complexity*) of a T.M.

A T.M. has time complexity $T(n)$ if it halts in at most $T(n)$ steps (accepting or not) for all input strings of length n.

Polynomial time: $T(n) = O(n^c)$ for some fixed c

(fixed means independent from n, i.e. the input-size)

Examples:

$$\left.\begin{array}{l} O(n^2) \\ O(n \cdot \log n) \\ O(n^{3.14}) \end{array}\right\} \text{polynomial time}$$

$$\left.\begin{array}{l} O(n^{\log n}) \\ O(2^n) \end{array}\right\} \text{non-poly}$$

Complexity theory considers *tractable* all problems with poly-time algorithms:

Motivations:

1) robustness wrt the computation model

all general computation models can simulate each other in poly-time ⇒ they define the same class of tractable probl.

2) robustness wrt combining algorithms

(a polynomial of a polynomial is still a polynomial)

3) going from polynomial to non-polynomial is drastic also in practice (e.g. compare $10 \cdot n^4$ with $0.1 \cdot 2^n$, when n grows)

4) Most practically used algorithms that are polynomial are so with a low coefficient (i.e. $T(n) = O(n^c)$, with c typically ≤ 3.

Time complexity classes:

Definition: $P = \{ L \mid L = \mathcal{L}(M) \text{ for some poly-time DTM } M \}$

$NP = \{ L \mid L = \mathcal{L}(N) \text{ for some poly-time NTM } N \}$

Note: both DTMs and NTMs must be halting T.M.s

From the definition we have immediately: $P \subseteq NP$
(every NTM is also a DTM)

Note: being in P corresponds to the intuition that the problem can be solved efficiently.

Instead, being in NP means intuitively that, given a solution, we can check efficiently whether it is correct.

Satisfiability:

Boolean formula: operands: $x_1, \dots, x_n$
operators: $\wedge, \vee, \neg$
formula $F(x_1, \dots, x_n)$

Satisfiability problem: given a boolean formula $F(x_1, \dots, x_n)$, is there a truth assignment (i.e., an assignment of true/false values) for $x_1, \dots, x_n$ that satisfies F (i.e., makes F evaluate to true)?

Example: $F(x_1, x_2) = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$

is satisfiable: $x_1 = 1$, $x_2 = 1$

$F(x_1, x_2) = x_1 \wedge (\neg x_1 \vee x_2) \wedge \neg x_2$

is not satisfiable

We first show how we can convert it to a language problem:

- we must encode formulas as strings

$\Sigma = \{ \wedge, \vee, \neg, (,), x, 0, 1 \}$

variable x_i: x ⟨i in binary⟩

e.g. x_5 is encoded as $x101$

$\Rightarrow$ we obtain that $F(x_1, \ldots, x_n)$ can be encoded as a string over Σ.

$L_{SAT} = \{ w \mid w$ encodes a satisfiable formula$\}$

Theorem: $L_{SAT} \in NP$ (i.e. satisfiability is in NP)

Proof:

It suffices to show a poly-time NTM N s.t. $\mathcal{L}(N) = L_{SAT}$

N runs in two steps:

1) "guess" a truth assignment F for $x_1, \ldots, x_n$

2) evaluate F on truth assignment and whether it has value true.

We have: F satisfiable $\Leftrightarrow \exists$ satisfying T.A.

$\Leftrightarrow N$ has accepting execution

Running time: step 1) $O(n)$

step 2) $O(n^2)$ with multiple tapes $\Rightarrow O(n^4)$

q.e.d.

Note: - All decision problems can be converted to language problems, by encoding the input as a string.

- We know that $L_{SAT} \in NP$, but we do not know whether $L_{sat} \in P$:
  - we cannot exploit the conversion NTM $\rightarrow$ DTM, since it causes an exponential blowup in running-time
- under the standard NTM - DTM conversion, the DTM will have to try all possible truth-assignments (2^{24})

In fact: open whether $L_{SAT} \in P$

Special case of SAT: CSAT

conjunctive normal form:

(note: we use $+$ for $\vee$
and $\cdot$ for $\wedge$)

- literal: variable x_i or its negation $\overline{x_i}$
- clause: sum/or of literals: $C_j = x_1 + \overline{x_2}$
- CNF-formula: product/and of clauses: $F = C_1 \cdot \ldots \cdot C_m$

Thus $F = \prod_{j=1}^{m} C_j$ with $C_j = \sum_{i=1}^{t_j} x_{ji}$

CSAT-problem: given a CNF formula F,
decide whether F is satisfiable

Since SAT $\in$ NP, we have also CSAT $\in$ NP k-CNF-formula: each clause has exactly k literals

1-SAT : $(\bar{x}_1)\cdot(x_2)\cdot(x_3)$

2-SAT : $(x_1+\bar{x}_2)\cdot(\bar{x}_1+x_2)$

3-SAT :

Facts: 1-SAT $\in$ P (trivial)

2-SAT $\in$ P (not so easy - via graph reachability)

3-SAT $\in$ P is still open

There are many (thousands) problems like SAT and CSAT that can be easily established to be in NP as follows:

Step 1 : "guess" some solution S

Step 2 : verify that S is a correct solution

Note: Step 1 exploits nondeterminism, and is clearly polynomial (running time of a NTM)

Step 2, for the problem to be in NP, must be carried out deterministically in poly-time (polynomial verifiability)

Examples:

- Travelling salesman problem (TSP)

input : - graph $G=(V,E)$ with edge lengths $l(u,v)$

- integer k

problem : does G have a tour (visiting each node exactly once) of length $\leq k$?

TSP $\in$ NP

Step 1 : guess a tour

Step 2 : check that length of tour is $\leq k$

- Clique : input - graph $G = (V, E)$
  - integer k

  problem: does the graph have a clique of size k

  (a clique is a subgraph of G in which each pair of nodes is connected by an edge)

- Knapsack : input : - set of items, each with an integer weight
  - capacity k of a knapsack

  problem : is there a subset of the items whose total weight matches the capacity k

This property explains why so <u>many practical problems are in NP</u>:
- problems ask for the design of mathematical objects (paths, truth assignments, solutions of equations, VLSI-routes, ...)
- sometimes we look for the best solution (or a solution that matches some condition) that matches the specification
- the solution is of small (polynomial) size, otherwise it would be useless
- it is simple (poly-time) to check whether it matches the spec.

but, there are exponentially many possible solutions

If we had $P = NP$, all these problems would have efficient (poly-time) solutions.

But we currently believe that $P \neq NP$.

<u>Assuming $P \neq NP$</u>, how do we determine which problems of NP are not in P (i.e., we know they don't have an efficient algorithm)?

18/1/2006

NP-completeness

Key idea: we define NP-completeness in such a way that if we show that an NP-complete problem is in P, then all problems in NP would be in P. (i.e., we would have P = NP)

It follows: assuming $P \neq NP$, an NP-complete problem cannot be in P

Poly-time reduction:

Problem X reduces to problem Y in poly-time ($X <_{poly} Y$) if there is a function R (the poly-time reduction) s.t.

1) $w \in L_X \iff R(w) \in L_Y$
2) R is computable by a poly-time DTM

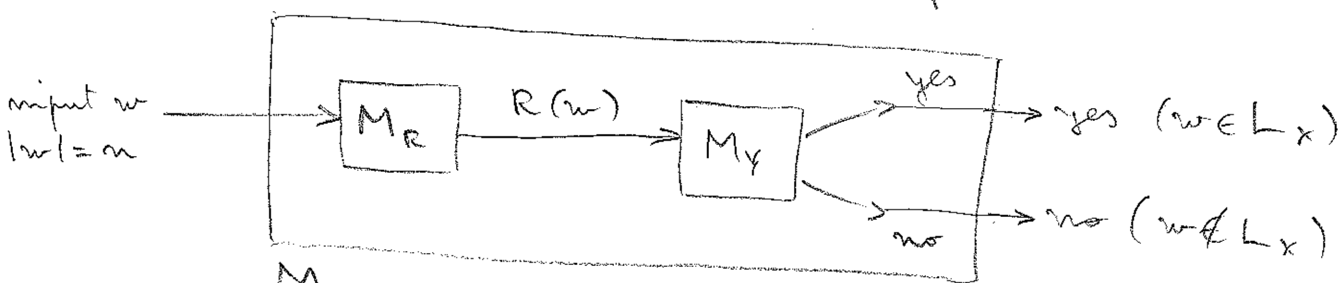
(L_X is the language encoding of problem X)

Theorem: $X <_{poly} Y$ and $Y \in P \Rightarrow X \in P$

Proof: let M_R be a poly-time DTM for R

M_Y — " — Y

We construct a DTM M_X for X as follows



M_X

Running time of M_X:

suppose: M_R runs in time $T_R(n) \leq n^a$

M_Y — " — $T_Y(n) \leq n^b$

Let $|w| = n$

Then $|R(w)| \leq n^a$

$\Rightarrow M_X$ runs in time

$$T_X(n) \leq T_R(n) + T_Y(T_R(n)) =$$
$$= n^a + (n^a)^b = O(n^{a \cdot b})$$

q.e.d.

Corollary: $X <_{poly} Y$ and $X \notin P \Rightarrow Y \notin P$

Definition: Problem Y (or language L_Y) is NP-hard if $\forall X \in NP$ we have $X <_{poly} Y$

Intuitively: an NP-hard problem is at least as hard as any problem in NP

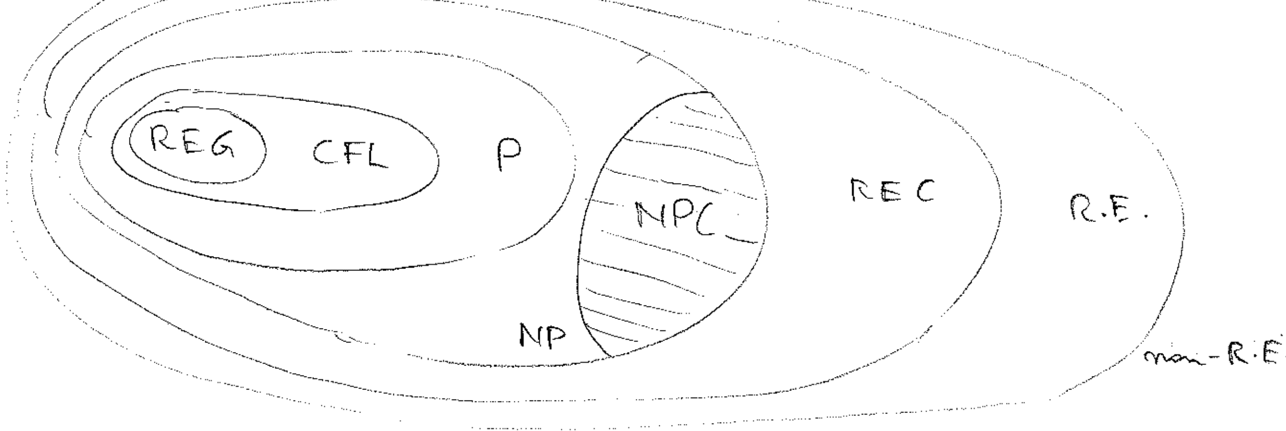
Immediate: Y is NP-hard and $Y \in P \Rightarrow P = NP$

Definition: Y is NP-complete if
1) $Y \in NP$ and
2) Y is NP-hard

Intuitively: NP-complete problems are the hardest problems in NP. If one of them is in P, then all problems in NP are in P.

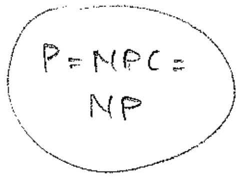
Hence: NP-completeness is a strong evidence of intractability.

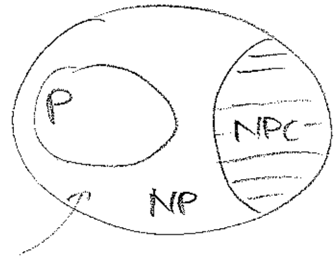
Languages:



Note: relationship between P, NPC and NP

either P = NP or P ≠ NP 10/12/2007





in this case we know there are problems in NP that are neither in P nor NPC (proof is complicated)

How do we prove problems to be NP-complete?

<u>Theorem</u>: X is NP-hard and $X <_{poly} Y \Rightarrow Y$ is NP-hard

Proof: $NP <_{poly} X <_{poly} Y$

But, to exploit this result, we need a first NP-hard problem:

<u>Cook's theorem</u>: CSAT is NP-hard 17/1/2007

Proof idea: we must show: $\forall L \in NP : L <_{poly} L_{CSAT}$

Fix $L \in NP$ and let M_L be a poly-time NTM for L.

We must show a poly-time reduction R_L:

input: string w

output: CNF formula $F = R_L(w)$ s.t.

$$w \in \mathcal{L}(M_L) \Leftrightarrow F \text{ is satisfiable}$$

Idea: F encodes the computation of M_L on w.

Suppose $w \in \mathcal{L}(M_L)$ and $|w| = n$.

then there exists a sequence of IDs of M_L:

$$ID_0 \vdash ID_1 \vdash \cdots \vdash ID_\tau$$

with $ID_0 = q_0 w$

ID_τ is an accepting ID (i.e. M_L is in a final state.)

$\tau \leq P(n)$ (since M_L is poly-time)

We assume that $\tau = P(n)$ by adding $ID_{\tau+1}, ID_{\tau+2}, \ldots, ID_{P(n)}$ same as ID_τ

Idea: encode computation as matrix X

TAPE $\longrightarrow$

$w = a_1 \cdots a_n$

TIME $\downarrow$

	1	2	3		n	$n+1$	$n+2$			$P(n)$
0	q_0/a_1	a_2	a_3	....	a_n	␢	␢	....	␢	␢
1	b_1	q_1/a_2	a_3	...	a_n	␢	␢	...	␢	␢
2	b_1	b_2	q_3/a_3	...	a_n	␢	␢	...	␢	␢
$P(n)$	c_1	c_2	c_3	...	c_n	c_{n+1}	c_{n+2}		q/c_1	$␢_2$

M cannot use more than $P(n)$ cells

x_{it} : contents of tape cell i in ID_t

except for composite symbol $\boxed{q/x}$ to denote state and head position

We have that $w \in \mathcal{L}(M_L)$ iff

a) the matrix X is properly filled in

b) row 0 is ID_0

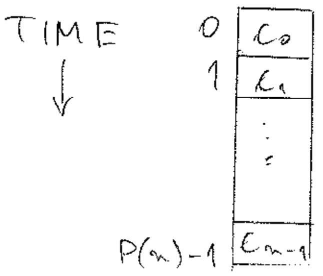
c) row $P(n)$ has final state

d) successive rows are related through legal transitions of M_L

M_L is NTM. Let k be the maximum degree of nondeterminism, i.e., for all q, X : $|\delta(q,X)| \leq k$.

To encode which of the possible transitions is chosen when going from ID_i to ID_{i+1} for the accepting sequence:

We use an array C of $P(n)$ elements (clue array)



$1 \leq c_i \leq k$

To represent X and C we use boolean variables

x_{itA} = true if cell i in ID_t contains A

c_{tl} = true if $c_t = l$

where $1 \leq i \leq P(n)$

$0 \leq t \leq P(n)$

$A \in \Gamma' = \Gamma \cup \underbrace{\Gamma \times Q}_{\text{composite symbols}}$

$1 \leq l \leq k$

Total number of variables is $O(P(n)^2)$, i.e., polynomial

To construct the CNF formula F we use 4 types of formulas (that are conjunctions of clauses)

type a) X and C are properly filled in:

UNIQUE(i,t): for each i and t, cell i in ID_t is uniquely filled

$$\left(\sum_{A \in \Gamma'} x_{itA}\right) \cdot \prod_{\substack{A,B \in \Gamma' \\ A \neq B}} \left(\overline{x_{itA}} + \overline{x_{itB}}\right)$$

UNIQUEC(t): for each t, $C[t]$ is uniquely filled

$$\Big(\sum_{l\in\{1,\dots,k\}} C_{t,l}\Big)\cdot \prod_{\substack{l,m\in\{1,\dots,k\}\\ l\neq m}} \left(\overline{C_{tl}}+\overline{C_{tm}}\right)$$

$\Rightarrow O(P(n)^2)$ clauses, which is still polynomial (since $1\leq i\leq P(n)$ and $0\leq t\leq P(n)$)

type b) $ID_0 = q_0 w = q_0 a_1 \cdots a_n$

INIT: $x_{1 0 \boxed{q_0/a_1}} \cdot x_{2 0 a_2} \cdots x_{n 0 a_n} \cdot$

$x_{n+1,0,\not b} \cdot x_{n+2,0,\not b} \cdots x_{P(n),0,\not b}$

$\Rightarrow O(P(n))$ clauses, each of length 1

type c) $ID_{P(n)}$ is accepting

ACCEPT: $\displaystyle\sum_{\substack{q\in F\\ A\in\Gamma\\ i\in\{1,\dots,P(n)\}}} x_{i,P(n),\boxed{q/A}}$

$\Rightarrow$ 1 clause of length $O(P(n))$

type d) legal transitions

consider ID_t and ID_{t+1}

t	A_1	A_2	....	q/A_j	A_{j+1}	..
$t+1$	B_1	B_2	...	B_j	p/A_{j+1}	

t $\boxed{C_t}$

In ID_{t+1} cell j depends only on 3 cells above it and on C_t

A_{j-1}	A_j	A_{j+1}
	B_j	

Various cases: (we assume that there are no stay moves)

1) A_{j-1}, A_j, A_{j+1} are not composite symbols

then $B_j = A_j$

2) A_{j-1} is $\boxed{q/X}$ and c_t'th move in $\delta(q,X)$ is (p,Y,R)

then $B_j = \boxed{p/A_j}$

3) A_j is $\boxed{q/X}$ and c_t'th move in $\delta(q,X)$ is $(p,Y,-)$

then $B_j = Y$

4) A_{j+1} is $\boxed{q/X}$ and c_t'th move in $\delta(q,X)$ is (p,Y,L)

then $B_j = \boxed{p/A_j}$

We use clauses that forbid illegal moves: LEGAL(t,j)

$$\prod_{D,E,F,G,H} \left(\overline{C}_{t,D} + \overline{X}_{j-1,t,E} + \overline{X}_{j,t,F} + \overline{X}_{j+1,t,G} + \overline{X}_{j,t+1,H} \right)$$

s.t. with class D and $\begin{array}{|c|c|c|} \hline E & F & G \\ \hline & H & \\ \hline \end{array}$ we have an <u>illegal</u> move

(NB. the illegal moves are those that do not correspond to 1–4 above)

$\Longrightarrow O(P(n)^2)$ clauses

(since $0 \leq t < P(n)$, $1 \leq j \leq P(n)$)

Formula F is the conjunction of all above clauses.
We can prove that $w \in \mathcal{L}(M_L)$ iff F is satisfiable.
It is easy to see that the reduction is poly-time q.e.d.

For a collection of NP-complete problems with discussion of variants see

Garey & Johnson.
Computers and Intractability. A guide to the Theory of NP-completeness
Freeman & Co. 1979

12/12/2007

coNP-completeness

Let us consider the complement of a problem in NP.

E.g. unsatisfiability

$$\text{UNSAT} = \{ F \mid F \text{ is a propositional formula that is not satisfiable} \}$$

Given a prop. formula F, how can we check whether $F \in \text{UNSAT}$?

- try all possible truth assignments for the vars in F
- if for none of these F evaluates to true, answer yes

Intuitively, this is very different from a problem in NP.

Note: in general, a NTM cannot answer yes to such a problem in polynomial time

<u>Definition</u>: $\text{coNP} = \{ L \mid \overline{L} = \Sigma^* \setminus L \in \text{NP} \}$

Note: many problems in coNP do not seem to be in NP.

We might conjecture NP $\neq$ coNP

This conjecture is stronger than P$\neq$NP.

- indeed, since P = coP, we have that NP $\neq$ coNP implies P $\neq$ NP
- but we might have P $\neq$ NP, and still NP = coNP

The following result shows a strong connection between NP-complete problems and the conjecture that NP $\neq$ coNP.

<u>Theorem</u>: If for some NP-complete problem/language L we have $\overline{L} \in NP$ (i.e., $L \in coNP$), then NP = coNP.

Proof: Assume $L \in NPC$ and $\overline{L} \in NP$.

1) We show NP $\subseteq$ coNP.

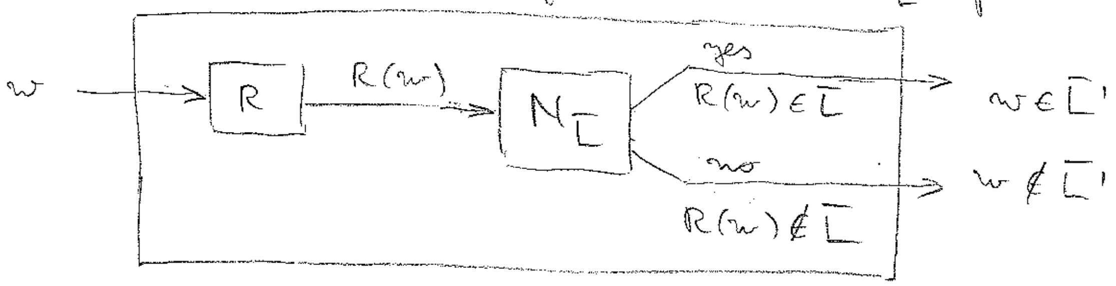
Let $L' \in NP$. We show $L' \in coNP$, i.e. $\overline{L'} \in NP$.

Since $\overline{L} \in NP$, there is a poly-time NTM $N_{\overline{L}}$ s.t. $\mathcal{L}(N_{\overline{L}}) = \overline{L}$.

Since $L' \in NP$ and $L \in NPC$, $L' <_{poly} L$, i.e. there is a polytime reduction R s.t.

$$w \in L' \iff R(w) \in L \qquad \text{i.e.}$$
$$w \in \overline{L'} \iff R(w) \in \overline{L}$$

We can construct a poly-time NTM $N_{\overline{L'}}$ for $\overline{L'}$

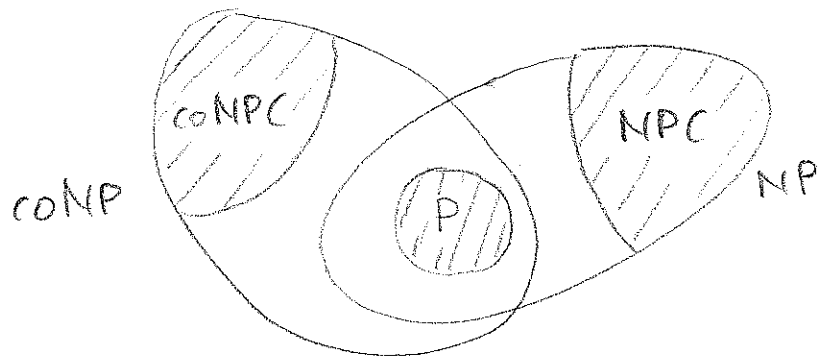


$N_{\overline{L'}}$

2) coNP $\subseteq$ NP. Similar

q.e.d.

We get the following picture (assuming $P \neq NP$
$NP \neq coNP$)



Note: it is not known whether $P = NP \cap coNP$