SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

FOUNDATIONS OF COMPUTER SCIENCE
LECTURE 15: NPC

Prof. Daniele Gorla

NP completeness

For certain **NP** problems their individual complexity is related to that of the entire class

→ If a polynomial time algorithm exists for any of these problems,
all problems in **NP** would be polynomial time solvable (hence, **P** = **NP**)

Important both theoretically and practically:

- If any problem in **NP** requires more than polynomial time, every **NPC** one does. Furthermore, to prove that **P** equals **NP**, you 'only' need to find a polynomial time algorithm for one **NPC** problem.
- The phenomenon of **NP**-completeness may prevent wasting time searching for a (non-existent) polynomial time algorithm to solve a particular problem

  → We believe that **P** is different from **NP**

  → Proving that a problem is **NPC** is a strong evidence of its non-polynomiality

To define **NPC**, we need to define what is

1. a «reduction» from one problem to another
2. when such a reduction is «efficient»

Polynomial reducibility

SAPIENZA
Università di Roma
Dipartimento di Informatica

DEFINITION

A function $f: \Sigma^* \longrightarrow \Sigma^*$ is a ***polynomial time computable function*** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

DEFINITION

Language A is ***polynomial time mapping reducible***, or simply ***polynomial time reducible***, to language B, written $A \leq_P B$, if a polynomial time computable function $f: \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

Prop.: If $A \leq_P B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.

Proof

Let M be the polynomial time decider for B and M' the polynomial time TM for the reduction f of A to B.

To decide A on w, we first run M' (to compute $f(w)$), then run M on its output and finally return whatever M returns. Q.E.D.

NPC

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Cor.: If $B \in \mathbf{NPC}$ and $B \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.

Prop.: If $B \in \mathbf{NPC}$ and $B \leq_P C \in \mathbf{NP}$, then $C \in \mathbf{NPC}$.

Proof

By hypothesis

- every $A \in \mathbf{NP}$ polynomially reduces to B (because $B \in \mathbf{NPC}$)
- B polynomially reduces to C.

Since the combination of polynomial reductions is a polynomial reduction, we have that every $A \in \mathbf{NP}$ polynomially reduces to C.

Since by hypothesis $C \in \mathbf{NP}$, we can conclude.

Q.E.D.

NPC proofs

To exploit the power of **NP**-completeness, we have two tasks to carry out:

1. Find a problem that is in **NPC**
   → this is very difficult !!

2. After this, to prove that a problem is in **NPC** it suffices to
   - Show that it is in **NP**
     → This is usually very easy (show that it can be polynomially verified)
   - Find a **NPC** problem that is polynomially reducible to it
     → this can be easy or not (but surely easier than 1.)

The first NPC problem (1)

SAPIENZA
Università di Roma
Dipartimento di Informatica

In boolean algebra, variables can take values in {0,1}.

The ***Boolean operations*** AND, OR, and NOT (represented by the symbols ∧, ∨, and $\overline{\cdot}$) are defined as:

$$
\begin{array}{lll}
0 \wedge 0 = 0 & 0 \vee 0 = 0 & \overline{0} = 1 \\
0 \wedge 1 = 0 & 0 \vee 1 = 1 & \overline{1} = 0 \\
1 \wedge 0 = 0 & 1 \vee 0 = 1 & \\
1 \wedge 1 = 1 & 1 \vee 1 = 1 &
\end{array}
$$

A ***formula*** is an expression involving constants, variables and operations.

A formula is ***satisfiable*** if some assignment of 0s and 1s to its variables makes the formula evaluate to 1.

The ***satisfiability problem*** is to test whether a Boolean formula is satisfiable or not.

As a language, we define

$$SAT = \{ \langle \varphi \rangle \mid \varphi \text{ is a satisfiable Boolean formula}\}$$

The first NPC problem (2)

<u>Thm</u> (Cook & Levin, 1971): *SAT* ∈ **NPC**.

Proof

Easily, *SAT* ∈ **NP**:

- Given a formula φ with *n* variables, a certificate is a sequence of *n* bits
- We verify that the given sequence is a satisfying assignment by:
  1. Replacing within φ every x_i with the i-th bit of the sequence
  2. Computing the final value and comparing it to 1
- Both operations are linear in the size of the formula (i.e., in the number of its variables and operators)

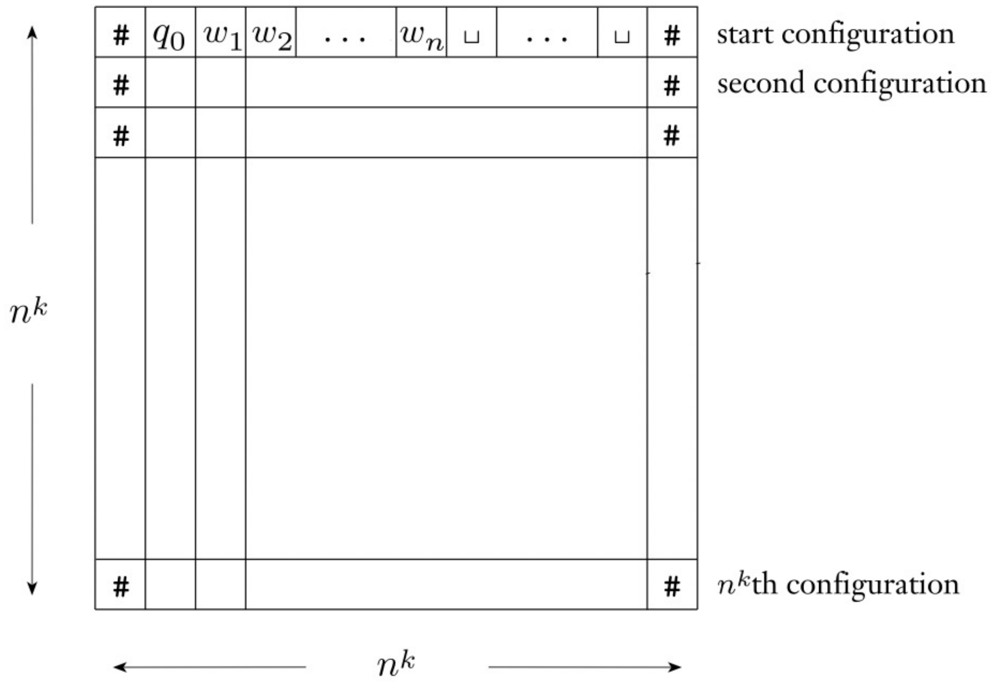
To show that every problem *A* ∈ **NP** polynomially reduces to *SAT*, we

- Consider a **NP** decider *N* for *A,* and
- For every *w*, we polynomially build a φ such that *N* accepts *w* IFF ⟨φ⟩ ∈ *SAT*.

Fix an *A* ∈ **NP** and a *w* ∈Σ*; let |*w*| = *n* and n^k–3 be (an upper bound on) the time spent by *N* to accept *w*.

The first NPC problem (3)

- A ***tableau*** for N on w is an $n^k \times n^k$ matrix whose rows are the configurations of a branch of the computation of N on input w.
- The first row of the tableau is the starting configuration of N on w.
- Each row follows the previous one according to N's transition function.
- For convenience, we assume that each configuration starts and ends with a # symbol; therefore, the first and last columns of a tableau are all #s.



- A tableau is ***accepting*** if any row of the tableau is an accepting configuration.
- N accepts w IFF there exists an accepting tableau for N on w.

The first NPC problem (4)

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Variables of formula φ:

- Say that Q and Γ are the states and tape alphabet of N
- Let $C = Q \cup \Gamma \cup \{\#\}$
- Each of the $(n^k)^2$ entries of a tableau is called a ***cell***.
- The cell in row i and column j is denoted $cell[i, j]$ and contains a symbol from C.
- For each $i, j \in \{1,\dots, n^k\}$ and $s \in C$, we have a variable $x_{i, j, s}$.
- We let $x_{i, j, s} = 1$ if and only if $cell[i, j]$ contains s.

The formula φ is the AND of four parts:

$$\varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{move}} \wedge \varphi_{\text{accept}}\,.$$

The first NPC problem (5)

φ_{cell} ensures that the assignment gives 1 to exactly one variable for each cell.

It is defined as
$$\bigwedge_{1\leq i,j\leq n^k} \bigvee_{s\in C}\left(x_{i,j,s} \wedge \bigwedge_{t\in C\setminus s} \overline{x_{i,j,t}} \right)$$

This formula should be understood as:

for all *i, j* there exists an *s* such that

1. $x_{i,j,s} = 1$ (i.e., *cell*[*i, j*] contains symbol *s*), and
2. for every $t \neq s$, we have $x_{i,j,t} = 0$ (i.e., *cell*[*i, j*] doesn't contain symbol *t*)

Indeed, a conjunction acts as a universal quantification: to satisfy $\varphi_1 \wedge \ldots \wedge \varphi_k$, all the φ_i must evaluate to 1.

Similarly, a disjunction acts as an existential quantification: to satisfy $\varphi_1 \vee \ldots \vee \varphi_k$, at least one of the φ_i must evaluate to 1.

Actually, this formula ensures that exactly one variable $x_{i,j}$ holds 1:

→ if $x_{i,j,s} = 1$, then $x_{i,j,t} = 0$ for $t \neq s$; so, every conjunct that starts with $x_{i,j,t}$ will be false.

The first NPC problem (6)

φ_{start} ensures that the first row of the table is the starting configuration of N on w:

$$x_{1,1,\#} \wedge x_{1,2,q_0} \wedge$$
$$x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge$$
$$x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

φ_{accept} guarantees that an accepting configuration occurs in the tableau

→ it ensures that q_{accept} appears in one of the cells:

$$\bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{accept}}$$

The first NPC problem (7)

φ_{move} guarantees that each row of the tableau corresponds to a configuration that legally follows the preceding row's configuration according to N's rules.

It does so by ensuring that each 2×3 window of cells is legal, where a 2×3 window is ***legal*** if that window does not violate the actions specified by N's transition function.

→ very heavy

EXAMPLE: Assume that $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$.

Then, the following are examples of legal windows:

a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	b	b
c	b	b

whereas the following are examples of not legal windows:

a	b	a
a	a	a

a	q_1	b
q_2	a	a

b	q_1	b
q_2	b	q_2

The first NPC problem (8)

According to the transition function of N, we have many 2×3 windows that are legal.

We number windows by the upmost central position:

	j-1	j	j+1
i			
i+1			

Hence, formula φ_{move} must require that every position i, j contains a 6-tuple of elements of $C = Q \cup \Gamma \cup \{\#\}$ that form a legal window.

So, formula φ_{move} is

$$\bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \ \bigvee_{\substack{a_1,\ldots,a_6 \\ \text{legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6} \right)$$

The first NPC problem (9)

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Time complexity:

1. Number of variables:

- the tableau is an $n^k \times n^k$ table, so it contains n^{2k} cells.
- Each cell has h variables associated with it, where $h = |C|$.
- Because h depends only on N and not on n, the number of variables is $O(n^{2k})$.
- REMARK: each variable has indices that range up to n^k; so they require $O(\log n)$ symbols to be codified into the formula

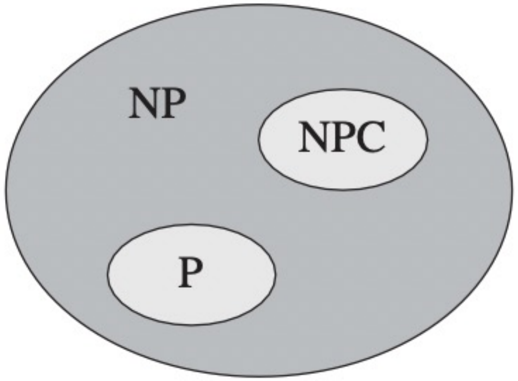
  → but $O(\log n) < O(n)$, so this factor does not influence polynomiality of the reduction

2. Size of the formula:

- φ_{cell} contains a fixed-size fragment of the formula for each cell of the tableau, so its size is $O(n^{2k})$.
- Formula φ_{start} has a variable for each cell in the top row, so its size is $O(n^k)$.
- Formulas φ_{move} and φ_{accept} each contain a fixed-size fragment of the formula for each cell of the tableau, so their size is $O(n^{2k})$.
- Thus, φ's total size is $O(n^{2k})$. <u>Q.E.D.</u>

NPC and P vs NP

As usual, the scenario is not known, but the most widely believed conjecture is



The typical NP problem that is not believed to be NPC is ***graph isomorphism***:

- Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are ***isomorphic*** if there exists a bijection f: $V_1 \longrightarrow V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v))$ E_2.
- *GRAPH-ISOM* is the problem of saying whether two graphs are isomorphic or not.
- *GRAPH-ISOM* is trivially in **NP** (a certificate is the bijection among the vertices)
- Nobody has been able to prove its **NP**-completeness
- By contrast, a related **NPC** problem is ***sub-graph isomorphism***:

  SUBG-ISOM: given two graphs, say whether the first one is isomorphic to a subgraph of the second one