Chapter 1

Complexity classes

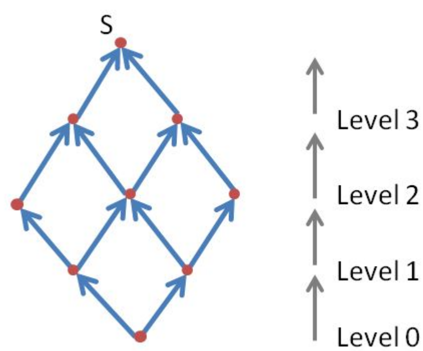
We have this expression:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$$

However, we do not know if these they are different or not.
Order on problems: $S \leq_p T$, where $\leq_p$ is the poly-time reduction.

Theorem 1.0.1. *There is a problem $S \in NP$ such that $\forall T \in NP : T \leq_p S$*

Proof. Not trivial, we could have:



with $\nearrow$:$\leq_p$: the element which point to the other is the simplest one. $\square$

Problem 1. SAT problem :

- Instance: A boolean formula, e.g. $(P \vee \neg Q) \wedge Q$, with $\vee$ which corresponds to "or", $\neg$: "not" and $\wedge$: "and".

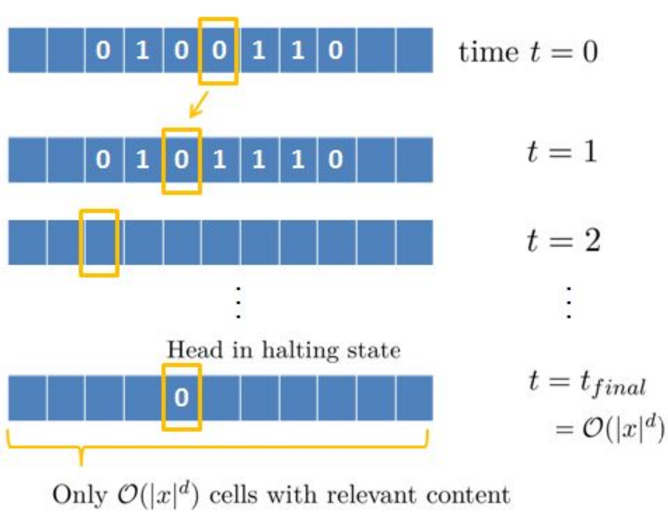
- Output: Yes, if exists true value for the variables so that the formula evaluates to true, e.g.
  - $(P \vee \neg\, Q) \rightarrow$ Yes, for P= true
  - $Q \wedge \neg\, Q \rightarrow$ No

Theorem 1.0.2. $\underbrace{SAT \in NP \quad and \quad \underbrace{\forall T \in NP : T \leq_p SAT}_{\text{"SAT is NP-hard"}}}_{\text{"SAT is NP-completed"}}$

Proof. (Sketch)

- SAT $\in$ NP: certificate= list of TRUE/FALSE values for all variables that make a formula evaluated to YES $\Rightarrow$ ok!
- $\forall T \in NP : T \leq_p$ SAT:
  Let T be in NP, then it exists a Turing Machine M such that $M(x, c) = YES$ for some poly-length certificate c, running in poly-time, if and only if x=YES instance of T.

  We detail the Turing Machine:



  $\Rightarrow \mathcal{O}\left(|x|^{2d}\right)$ variables $P_{t,m}$=TRUE if cell m at time t is 1, with t, the time and m, the space. $\mathcal{O}\left(|x|^d\right)$ boolean variables describing the head state $P_{t,s}$=TRUE if and only if Head is in state s at time t. $\mathcal{O}\left(|x|^d\right)$ variables $Q_{t,m}$=TRUE if and only if head is reading cell m at time t.

We can describe the rules of the Turing Machine by a boolean formula: e.g. t, m, the transition $s_1 \longrightarrow s_2$ if we read a 0. (If P_{t,s_1} and $\neg P_{t,m}$ and $Q_{t,m}$ then P_{t+1,s_2}).

(Reminder: " If A then B" $\Leftrightarrow \neg A \vee B$)

The whole computation is encoded into a big, but poly-size (and computed in poly-time), boolean formula $\phi_{x,c,M}$ (described by $P_{t,m}$ for $t = 0 \equiv$ Initial state).

The question "$\exists ? c : T(x,c) = YES$" (for a given x).

Amounts: "Are there truth values for the $P_{0,m}$ encoding c such that $\phi_{x,c,T}$=YES ?", with x and T which are given and c that to be found.

$\Rightarrow$ This is a SAT-instance

$\Rightarrow T \leq_p$ SAT

□

Intuition: SAT is "the hardest" problem in NP

The main technique to prove that a problem S is NP-complete is:

- prove that S $\in$ NP
- for some other NP-complete problem S_0, e.g. S_0 =SAT, prove that $S_0 \leq_p S$

You can conclude:

- S is NP-complete
- $S \equiv_p S_0 \equiv$ SAT

Since we conjecture that $P \neq NP$, the practical consequence is that we cannot reasonably hope to find a poly-time algorithm for S.

To this day, thousands of interesting problems have been proved NP-complete.

Example 1. *NP-complete problems:* 3 *(possibly negated) variables, related by* 1*.*

Problem 2. 3-SAP:

- Instance: A boolean formula in the form $\underbrace{(P \vee \neg Q \vee R)}_{\text{"}clause\text{"}} \wedge (\neg P \vee \neg R \vee S) \wedge$ $(P \vee P \vee Q)$. The clauses are linked by $\neg$.
- Output: YES if satisfiable (i.e. true for some value of $P, Q, R, S, \cdots$)

Theorem 1.0.3. *3-SAT is NP-complete*

Proof.
- 3-SAT $\in$ NP: clear
- 3-SAT is NP-hard ?, we prove SAT $\leq_p$ 3-SAT

$\square$

The reduction proceeds by transforming in poly-time every boolean formula into the normal form $(\vee \ \ \vee) \wedge (\vee \ \ \vee) \wedge \cdots$ using the identities of boolean logic in a systematic way. e.g. :
$(P \wedge Q) \vee R \equiv (P \vee R) \wedge (Q \vee R)$ distribution
$\neg(P \vee Q) \equiv \neg P \vee \neg Q$ (de Morgan) and More.

3-SAT can help prove that even more problems are NP-complete.

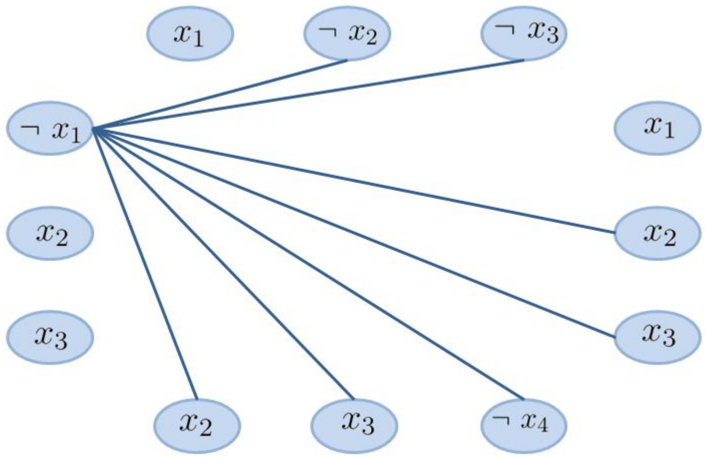
Theorem 1.0.4. *CLIQUE is NP-complete*

Proof.
- CLIQUE $\in$ NP: certificate of a YES-instance= list of nodes in the CLIQUE
- CLIQUE is NP-hard ?
  we show 3-SAT $\leq_p$ CLIQUE.
  We have to transform every formula $\phi = \underbrace{(\cdots \vee \cdots \vee \cdots) \wedge (\cdots) \wedge (\cdots)}_{\text{k clauses}}$ into a CLIQUE instance: a graph or an integer.

  We illustrate on an example:
  $\phi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \neg x_4)$: 4 clauses
  $f(\phi) =$(Graphe,4)=instance of CLIQUE

  All pair of nodes are linked except:

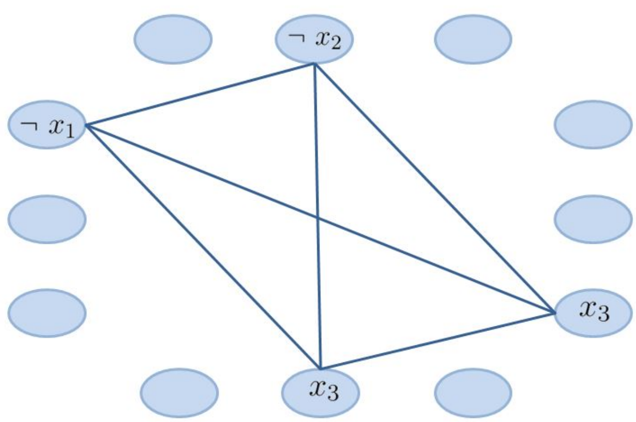
  - inside a triplet representing a clause
  - x_i with $\neg x_i$



We check, if ϕ is satisfiable then it exists k-clique in Graph. Indeed, if ϕ satisfiable by:

$\neg x_1$	=TRUE in clause	1
$\neg x_2$	=TRUE in clause	2
x_3	=TRUE in clause	3
x_4	=TRUE in clause	4

(at least one is true in every clause)



If exists k-CLIQUE in a graph then ϕ is satifiable. Indeed, we find clique then we can see that the assignment :

$$\begin{aligned} \neg x_1 &= \text{TRUE} \\ \neg x_2 &= \text{TRUE} \\ x_3 &= \text{TRUE} \\ x_3 &= \text{TRUE} \end{aligned}$$

(and anything for other variables) makes Φ true (because well defined and makes one possibly negated) variable true in every clause.

□

Often practical problems are optimization problems.

Problem 3. MAX clique:

- Instance: A graph
- Output: Clique of max size

We convert this into a decision problem: CLIQUE and we can run CLIQUE for several k (e.g.by dichotomy)