Complexity

We discussed several algorithms for decidable problems.

Example:

The membership problem for context-free grammars ("given a CFG G and a string x, is x in L(G)?")

We gave an algorithm which took about $|P|^2n$ steps (it was trying all possible derivations of length 2n for a grammar in the Chomsky normal form and $|x|=n$).

We also said that there exists an algorithm (Cocke-Younger-Kasami, also known as CYK) that takes about $n^3$ steps.

Which algorithm is better?  

\textit{Second (faster)}
We say that a language (problem) \( L \) is **tractable** if there exists a constant \( c \) and a TM \( T \) such that \( L(T) = L \) and the computation of \( L \) on input \( x \) always halts after at most \( |x|^c \) steps.

**Class P** contains all tractable languages.

(P stands for “polynomial-time.”)

**Remark**: we are not restricted to Turing machines. If we solve a problem in, e.g., Java, then there is a TM which follows the same computation and it is slower by only a polynomial factor.
Class $P$ and other classes

Examples of problems in $P$:
- Check if balanced parenthesis
- Graph algo (see class $BQP$) < does there exist a path from $A$ to $B$?

Is every problem in $P$?
- No, e.g. halting problem

Is every decidable problem in $P$?
- There are problems in $Rec$ which are provably not in $P$
**Satisfiability**

Input: a CNF (conjunctive normal form) formula $\phi$

Output: YES, if $\phi$ is satisfiable (i.e. it is possible to assign true/false values to the variables in $\phi$ so that $\phi$ is true), and NO otherwise.

I.e., SAT is the set of all satisfiable CNF formulas.

Example:

$$(x_1 \lor \neg x_2) \land (x_2 \lor x_3 \lor \neg x_4) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4)$$

Is SAT in P?

*nobody knows*
Class NP

A nondeterministic polynomial-time algorithm for SAT?

Example:

\[ q = (x_1 \lor \neg x_2) \land (x_2 \lor x_3 \lor \neg x_4) \land (\neg x_1 \lor \neg x_3 \lor \neg x_4) \]

1) nondeterministically guess true/false value for each variable \( \leftarrow \) poly-time

2) verify the guess, i.e. see if \( q \) is true \( \leftarrow \) poly-time
**Class NP**

**Class NP** contains languages \( L \) for which there is a NTM \( T \) such that \( L(T) = L \) and each computation of \( T \) halts after a polynomial-number of steps.

**Examples:**

**INDEPENDENT SET**

Input: a graph \( G \) and a number \( k \)

Output: YES if \( G \) contains \( k \) vertices such that no pair of these vertices is connected by an edge (an independent set of size \( k \))

**Claim:** INDEPENDENT SET is in NP.
**Class NP**

Class NP contains languages L for which there is a NTM T such that \( L(T) = L \) and each computation of T halts after a polynomial-number of steps.

**Examples:**

**VERTEX COVER**

Input: a graph \( G \) and a number \( k \)

Output: YES if \( G \) contains \( k \) vertices such that every edge is “covered” by at least one of these vertices (a vertex cover of size \( k \))

**Claim:** VERTEX COVER is in NP.
P vs. NP

We know that SAT, VERTEX COVER, and INDEPENDENT SET are all in NP (and we do not know if they are in P).

A BIG open problem: is $P = NP$? 

most people believe $P \neq NP$

What can we say about $P$ vs. $NP$?

$P \subseteq NP$

most people believe $P \neq NP$

$NP$-complete
Reducibility

We say that a problem $L_1$ can be \textbf{polynomial-time many-one reduced} to a problem $L_2$ if we can construct a polynomial-time algorithm for $L_1$ which uses a single call of a polynomial-time algorithm for $L_2$ as a subroutine. We write $L_1 \leq_p L_2$.

\textbf{Example} : INDEPENDENT SET $\leq_p$ VERTEX COVER

\begin{itemize}
  \item $V$ - set of all vertices
  \item $S \subseteq V$ - an indep. set then $V-S$ is a vertex cover (and vice versa)
  \end{itemize}

Suppose we have $\text{VertexCover}(G, k)$

Then we can:

\begin{verbatim}
IndepSet$(G, k)$ {
  return $\text{VertexCover}(G, |V| - k)$,
}
\end{verbatim}

Thus $\text{IND. SET} \leq_p \text{VERTEX COVER}$
A problem $L$ in NP is **NP-complete** if every other problem in NP can be reduced to $L$.

(I.e., NP-complete problems are the hardest in NP.)

**Cook's Thm:** SAT is NP-complete.

A possible attack on “$P=NP$?”:

*Give a polynomial-time algorithm for an NP-complete problem. Then $P=NP$.*
NP-completeness

Many real-life NP-complete problems, e.g., check List of NP-complete problems on Wikipedia.

How to prove that a problem is NP-complete?

- reduce from another NP-complete problem
  (e.g. if we know that INDEP. SET is NP-complete and we've shown \( \text{INDEP. SET} \leq_{p} \text{VERTEX COVER} \), \( \text{VERTEX COVER} \) must be NP-complete as well)

If my problem is NP-complete, what do I do now?

- approximation, heuristic, change of requirements