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? CYK
Class P

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 (for all sufficiently long inputs $x$).

**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:

- decide if a string is of even length
- decide if an arithmetic expression (using numbers, +, -, *) is valid
- decide if a number is prime \( \text{proved only lately} \)

Is every problem in P? **No**

  e.g. Halting

Is every decidable problem in P? **No**

  3 problems requiring an exponential \# steps
  \( \uparrow \) provably
Satisfiability

Input: a CNF (conjunctive normal form) formula \( \varphi \)

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

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

Example:

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

Is SAT in P?

\[
\begin{array}{c|c|c}
\text{variable} & \text{true} & \text{false} \\
\hline
x_1 & x_2 & x_3 & x_4 \\
\end{array}
\]

nobody knows


Class NP

A nondeterministic polynomial-time algorithm for SAT?

Example:

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

1) nondeterministically guess true/false value for each variable
   (takes \(\approx k\) steps for \(k\) variables)

2) verify that \(\varphi\) is true
**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 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.

Proof:
1) guess $k$ vertices (random)
2) verify that all edges are covered
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 that $P \neq NP$.

What can we say about P vs. NP?

$P \subseteq NP$
Reducibility

We say that a problem \( L_1 \) can be **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 \( L_1 \leq_p L_2 \).

**Example**: \( \text{INDEPENDENT SET} \leq_p \text{VERTEX COVER} \)

is saying:

if we had a poly-time algo for \( \text{VERTEX COVER} \),
then we could solve \( \text{INDEP. SET} \) in poly-time
**NP-completeness**

A problem \( L \) in \( \text{NP} \) is **NP-complete** if every other problem in \( \text{NP} \) can be reduced to \( L \).

(I.e., NP-complete problems are the hardest in \( \text{NP} \).)

**Cook’s Thm:** \( \text{SAT} \) is NP-complete.

A possible attack on “\( \text{P}=\text{NP?} \)”:

Give a polynomial-time algorithm for an NP-complete problem. 

then \( \text{P}=\text{NP} \)
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 another known NP-complete problem to it

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

approximate, heuristics, change requirements