CS 341 W22 Lecture 19
NP and NP-Complete

R. Peng, University of Waterloo
Recall

- We will study difficult problems by reducing them to each other
- For reductions, we prefer to work with decision problems
- P: class of decision problems with polynomial time algorithms
- $X \leq_p Y$: $X$ reduces to $Y$ in polynomial time
  - $X$ `is easier’ than $Y$
  - An algorithm for $Y$ can be used to solve a problem in $X$
Non-Deterministic Polynomial Time (NP)

• Decision problem
• If answer is yes, there is a succinct certificate that the answer is YES

Examples:
• Anything in P
• INDEPENDENT-SET: does G have an independent set of size $\geq k$. Can convince you such a set exists by providing S.
• Hamiltonian path/cycle: HAM(G) does G have a Hamiltonian path/cycle. Certificate is the path / cycle.
• Travelling salesperson problem: TSP(G, k) does G have a tour of cost $\leq k$. Certificate is the cycle.
Verification Algorithm

• Takes input and certificate and checks it

\[ A(x, y) \text{ is verification algorithm for decision problem } X(x) \text{ if} \]
• A takes two inputs, x (same input as X), and y (certificate)
• \( X(x) = \text{whether there exists some } y \text{ such that } (A(x, y) = \text{True}) \)

• A is a polynomial time verification algorithm if A runs in polynomial time, and length(y) \( \leq \text{poly(length(x))} \)
• Formal definition of NP (Non-deterministic Polynomial time): the class of decision problems that have polynomial time verification algorithms
Subset Sum $\in$ NP

- SUBSET-SUM($w[1...n]$, $W$): is there subset $S \subseteq [1..n]$ such that
  $$\sum_{i \in S} w[i] = W$$

- Verification algorithm $A(w[1...n], W, S)$:
  - $S$: certificate, the subset
  - Algorithm is just check the sum of $w[S]$, and return true if it’s same as $W$
TSP $\in$ NP

- TSP(G, k): does G have a tour of cost $\leq k$
- Verification algorithm A(G, k, p):
  - p: permutation of vertices
  - Algorithm:
    - Check if p is a valid permutation.
    - Check whether $(p[i], p[i+1])$ is an edge (with $p[n+1]$ overloaded to $p[1]$)
    - Sum total weights of these edges, check if it’s at most k
  - p has poly size, checking takes polytime
A few remarks

• Can usually arrive at certificates by proving the ‘solution’ the problem is looking for has poly-bounded size

• Showing formally $P \subseteq NP$: certificate is empty, verification algorithm is just the poly-time algorithm

• Problems that’s unclear whether they are in $NP$:
  • Unique subset sum: how to prove that a set is unique?
  • Steiner tree: connect points in plane (using possibly extra points) with lines of total length $\leq k$: it’s difficult to check if a sum of square roots is $\leq k$. 
coNP

- The class of decision problems where NO instances can be verified in poly time

X(x) is in coNP if there is algorithm A(x, y) such that
X(x) = False iff whether there exists some y such that (A(x, y) = True)

- Example: PRIME(x) ∈ coNP:
  - Certificate: numbers a and b ≥ 2 with ab = x

Aside: [AKS `02] PRIME(x) ∈ P
P, NP, coNP

What we do know:
• $P \subseteq NP$, $P \subseteq coNP$
• Any problem in NP, or coNP, can be solved in time $2^{poly(n)}$ by trying all certificates

Open
• $P =? NP$
• $NP =? coNP$
• $P \neq NP \cap coNP$

Possibilities (we don’t know which!)
Lecture 19 Part 1 Summary

• Definition of NP, coNP
• Verification algorithms and examples
• What you should know from Lecture 19 Part 1:
  • How to prove that a problem is in NP (certificate, verifier)
  • How to prove a problem is in coNP
• Next
  • NP-Complete Problems
NP Completeness

**Definition** a decision problem X is **NP-Complete** if
- \( X \in \text{NP} \)
- for every Y in NP, \( Y \leq_p X \)

i.e. X is (one of) the hardest problems in NP

- If X can be solved in polynomial time, than everything in NP can be solved in polynomial time.
- If X cannot be solved in polynomial time, then no NP-complete problem can be solved in polynomial time.
- Needs proof: \( X \in \text{coNP} \), then \( \text{NP} = \text{coNP} \)...
Showing X is NP-Complete

• First one is hard: need to some ANY Y in NP reduces to X (Y \leq_p X)
• Subsequent proofs easier: \leq_p is transitive: X \leq_p Y and Y \leq_p Z imply X \leq_p Z

\begin{align*}
Y_1 \leq_p & \quad Y_2 \\
Y_2 \leq_p & \quad X \\
Y_3 \leq_p & \quad X \\
X \leq_p & \quad Z
\end{align*}

• So to prove Z is NP-complete, just need to show: X \leq_p Z for SOME NP-complete X
NP-Completeness Proofs

To show a decision problem Z is NP-complete:

• Prove Z is in NP
• Prove $X \leq_p Z$ for a known NP-complete problem $X$
SAT

(later) First NP-complete problem: circuit SAT
(also later) Second NP-complete problem: SAT:

• Formula made of:
  • Boolean variables $x_1...x_n$
  • Logical operands: $\land$ “AND”, $\lor$ “OR”, $\neg$ “NOT”

• Problem: is there assignment of true/false to the variables so that the formula evaluates to true?

• E.g. $\neg(x_1 \land x_2) \lor (x_3 \land (x_5 \land \neg x_4))$

• Exercise: show that SAT is in NP
CNF and 3-SAT

- (will show later) SAT is NP-complete
- Even if it’s in conjunctive normal form (CNF)
  - Formula is AND of clauses
  - Each clause is OR of literals ($x_i$ or $\neg x_i$)

$$\left( x_1 \lor \neg x_2 \lor x_3 \right) \land \left( \neg x_1 \lor x_4 \right) \land \left( x_3 \lor x_4 \lor \neg x_5 \right)$$

- 3-SAT is also NP-complete:
  - Input: Boolean formula that’s an AND of clauses, each is the OR of 3 literals, each literal is a variable or its negation
  - Output: whether the formula can be made True by an assignment of the variables
- Aside: 2-SAT is in P
Lecture 19 Part 2 Summary

• Definition of NP-complete problems
• First NP-complete problems: SAT, 3-SAT
• What you should know from Lecture 19 Part 2:
  • The two steps for showing a problem is NP-complete
• Next
  • NP-completeness proofs
Independent Set

**Input**: graph G, value k

**Question**: does G have independent set of size $\geq k$

**Theorem** Independent-Set is NP-complete

**Proof**

- Independent set is in NP: a set S can be verified linear time
- $3$-SAT $\leq_p$ Independent Set

Suppose we have a polynomial time algorithm for independent set, use it to give a polynomial time algorithm for $3$-SAT
Idea

3-SAT

**Input:** 3-SAT formula $F$ with variables $x_1...x_n$, clauses $c_1...c_m$

**Question:** whether $F$ is satisfiable

- Build a graph $G$, and pick a number $k$ such that $G$ has independent set of size $\geq k$ if and only if $F$ is satisfiable

This lets us return $\text{IndependentSet}(G(F), k(F))$

Need to also prove that $G$ & $k$ are constructible in poly-time

This is called a many-one (one-shot) reduction.
Construction Idea

Equivalent view of 3-SAT

- Pick a literal from each clause (the one that’s satisfied):
- So we don’t simultaneously pick $x_i$ and $\neg x_i$: connect all pairs of $x_i$ and $\neg x_i$

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

One literal per clause:
- Add in-clause edges
- Set $k \leftarrow n$
Formal Construction

- For clause $C_i$, make a triangle, one vertex per each literal
- Add edges between conflicting literals
- Set $k \leftarrow m$

Correctness:
- If there is satisfying assignment, then choose one true literal per clause, they give independent set of size $m$
- If there is independent set of size $m$, there is exactly one per triangle. Set corresponding literals to true gives the solution to $F$

Runtime: $O(m^2)$, because number of vertices is $3m$. 

\[ (x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor \neg x_2 \lor x_3) \]
Form of Many-One Reduction

Formal definition: a many-one reduction $X \leq Y$ uses the algorithm for $Y$ once and outputs its answer.

Typical form:

- Assume we have algorithm for $Y$, $A_Y$
- Build algorithm for $X$:
  - Take input $x$, convert it to input for $Y$, $y$
  - Run $A_Y(y)$, return answer

- Correctness: $X(x)$ is YES if and only if $Y(y)$ is YES
- Polynomial time: prove constructing $y$ takes polynomial time
Lecture 19 Summary

• Definition of NP, coNP, NP-completeness
• First NP-completeness proofs
• What you should know from Lecture 19:
  • How to prove a problem is in NP/coNP
  • Definition of SAT and 3-SAT
  • Polynomial time many-one reduction
  • How to prove Z is NP-complete:
    • Prove Z is in NP
    • Show $X \leq_p Z$ (using a many-one reduction) for some NP-complete $X$
• Next: More NP-completeness proofs