

Toward NP-Completeness: Introduction

Almost all the algorithms we studied so far were bounded by some **polynomial** in the size of the input, so we call them **efficient** algorithms.

However, there are many problems for which **no polynomial-time** algorithm is known. We strongly suspect that many problems cannot be solved efficiently.

Toward NP-Completeness: Introduction

In the last few weeks, we have been dealing with several “**hard**” problems, like:

- Graph k -coloring: K^n possible color assignments
- Knapsack problem
- Graph Hamilton Circuit
- Bin-Packing (recall knapsack)
- Euclidean TSP (Traveling Salesman Problem)

Toward NP-Completeness: SAT

Satisfiability (SAT) problem:

- Conjunctive normal form (CNF): Let S be a Boolean expression in CNF. That is, S is the product (and) of several sums (or).
 - For example, $S = (x + y + z) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + \bar{z})$ where addition and multiplication correspond to the *and* and *or* Boolean operations, and each variable is either 0 (false) or 1 (true).
 - A Boolean expression is said to be **satisfiable** if there exists an assignment of 0s and 1s to its variables such that the value of the expression is 1.

Toward NP-Completeness: SAT

- Example: $(x + y + z) \cdot (\bar{x} + y) \cdot (x + \bar{z}) \cdot (z + \bar{y}) \cdot (\bar{x} + \bar{y} + \bar{z})$

At least one is true All three the same At least one is false
- Can x, y, z be set so that this expression is true? (NO, in the above case.)
- The SAT problem is to determine whether a given expression is satisfiable (without necessarily finding a satisfying assignment). 2^n different truth assignments.

Toward NP-Completeness: Preliminary

- **Decision problem:** problems with answer either “yes” or “no”. (e.g. Is there any TSP tour on graph G has length $\leq k$?)
- Decision problem can be viewed as **language-recognition problem:** (e.g. Is $(G, k) \in \text{TSP}$?)
 - U : the set of possible inputs to the decision problem.
 - $L \subseteq U$: the set of inputs which yield “yes”
 - L : the language corresponding to the problem.

Toward NP-Completeness: Preliminary

- **Definition:** Let L_1 and L_2 be two languages from the input spaces U_1 and U_2 . We say that L_1 is **Polynomially reducible** to L_2 \exists a polynomial-time algorithm that converts each input $u_1 \in U_1$ to another input $u_2 \in U_2$ such that $u_1 \in L_1$ iff $u_2 \in L_2$.

Note: The algorithm is polynomial in the size of the input u_1 . We assume that the notion of size is well defined in the input spaces U_1 and U_2 , so, in particular, the size of u_2 is also polynomial in the size of u_1 .

Toward NP-Completeness: Preliminary

- **Theorem:** If L_1 polynomially reducible to L_2 and \exists polynomial-time algorithm for L_2 , then \exists a polynomial-time algorithm for L_1 . (Denoted as $L_1 \leq_p L_2$.)
- **Corollary 1:** $L_1 \leq_p L_2$, and $L_2 \leq_p L_3 \implies L_1 \leq_p L_3$
- **Corollary 2:** $L_1 \leq_p L_2$, and $L_2 \leq_p L_1 \implies$ *polynomially equivalent*.

Note that:

- Problems we mentioned above are “non-solvable”. (Roughly speaking, “solvable” means \exists a polynomial-time algorithm.)
- They are *equivalent*: solve one \implies solve all.

Toward NP-Completeness: Preliminary

- **Definition:** X is **poly-time reducible** to Y **in sense of Turing**, written $X \leq_p^T Y$, if there exists an algorithm for solving X that would be polynomial if we took no account of the time needed to solve arbitrary instances of Y .
- **Example:**
 - X : Smallest_Factor ≥ 2 ?
 Y : Prime?
 Claim: $Y \leq_p^T X$
 - Z : # Distinct_Factors?
 Claim: $Z \leq_p^T X$

Toward NP-Completeness: Definitions and Classifications

Class of Decision Problems:

- **P:** Problems could be solved by *deterministic* algorithm in polynomial time.
- **NP:** Problems for which \exists a *non-deterministic* algorithm whose running time is a polynomial in the size of the input.
 - “small” solutions in poly-size. (like “certificate”)
 - Solution could be easily checked in poly-time.
- Non-deterministic poly-time algorithm.

Note: Whether $P = NP$ is not known, but most people believe $P \neq NP$.

Toward NP-Completeness: Definitions and Classifications

- **NP-Hard:** A problem X is called an NP-hard problem if **every** problems in NP is **polynomially reducible** to X .
- **NP-Complete:** A problem X is called an NP-complete problem if:
 - 1) X belongs to NP, **and**
 - 2) X is NP-hard.
- Also, X is **NP-complete** if $X \in NP$ **and** Y is **polynomially reducible** to X for some Y that is NP-complete.
- NP-Complete problems are the **hardest** problems in NP.

Toward NP-Completeness:

- **Cook's theorem:**
 The SAT problem is NP-complete.
- Once we have found an NP-complete problem, proving that other problems are also NP-complete becomes easier.
- Given a new problem Y , it is sufficient to prove that Cook's problem, or any other NP-complete problems, is **polynomially reducible** to Y .

NP-Completeness Proof

The following five problems are NP-complete: vertex cover (VC), dominating set (DS), 3SAT, 3-coloring, and clique.

Definition:

□ A vertex cover of $G=(V, E)$ is $V' \subseteq V$ such that every edge in E is incident to some $v \in V'$.

- **Vertex Cover (VC):** Given undirected $G=(V, E)$ and integer k , does G have a vertex cover with $\leq k$ vertices?
- **CLIQUE:** Does G contain a clique of size $\geq k$?

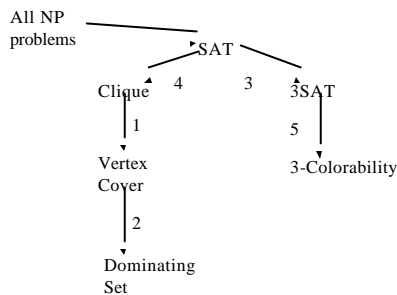
NP-Completeness Proof

- A dominating set D of $G=(V, E)$ is $D \subseteq V$ such that every $v \in V$ is either in D or adjacent to at least one vertex of D .
- **DS:** given G, k , does G have a dominating set of size $\leq k$?
- **3SAT:** Give a Boolean expression in DNF such that each clause has *exactly* 3 variables, determine satisfiability.

NP-Completeness Proof: Reduction

- To **prove NP-completeness** of a new problem, we must first prove that the problem belongs to NP, which is usually (but **not** always!!!) easy, then reduce a known NP-complete problem to our problem in polynomial time.
- The reduction order used for the five problems mentioned above is illustrated in the figure below. To make them easier to understand, we present the proofs in **order of difficulty** rather than the tree order. This order is indicated in the figure by the numbers of the edges.

NP-Completeness Proof: Reduction

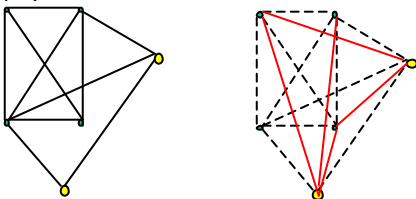


NP-Completeness Proof: Vertex Cover(VC)

- **Problem:** Given undirected $G=(V, E)$ and integer k , does G have a vertex cover with $\leq k$ vertices?
- **Theorem:** the VC problem is NP-complete.
- **Proof:** (Reduction from CLIQUE, i.e. given CLIQUE is NP-complete)
 - **VC is NP.** This is trivial since we can guess a cover of size $\leq k$ and check it easily in poly-time.
 - **Goal:** Transform arbitrary CLIQUE instance into VC instance such that CLIQUE answer is “yes” iff VC answer is “yes”.

NP-Completeness Proof: Vertex Cover(VC)

- **Claim:** CLIQUE(G, k) has **same** answer as VC ($\bar{G}, n-k$), where $n = |V|$.
- **Observe:** If $C = (u, F)$ clique in G , then $v-u$ is VC in \bar{G} .



NP-Completeness Proof: Vertex Cover(VC)

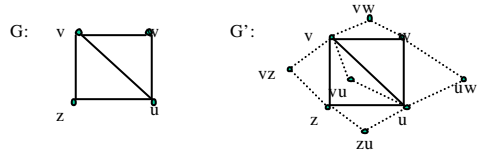
- **Observe:** If D is a VC in \bar{G} , then \bar{G} has no edge between vertices in $V-D$. So, we have k -clique in $G \Leftrightarrow n-k$ VC in \bar{G}
- Can transform in *polynomial time*.

NP-Completeness Proof: DS

□ Definition: A dominating set D of $G=(V, E)$ is $D \subseteq V$ such that every $v \in V$ is either in D or adjacent to at least one vertex of D .

- **Problem:** given G, k , does G have a dominating set of size $\leq k$?
- **Theorem:** DS is NP-complete
- **Proof:** (reduction from VC)
 - DS \in NP
 - Given instance (G,k) of VC, make every edge of G into a triangle and answer DS.

NP-Completeness Proof: DS



- G' has DS D of size k iff G has VC C of size k .
 - Without loss of generality, assume $D \subseteq V$, D has a DS \Rightarrow every edge hits some vertex in $D \Rightarrow D$ is a VC in G
 - C is a VC \Rightarrow new and old vertices dominated.

NP-Completeness Proof: 3SAT

- **3SAT:** Give a Boolean expression in DNF such that each clause has *exactly* 3 variables, determine satisfiability.
- **Theorem:** 3SAT is NP-Complete
- **Proof:**
 - 3SAT \in NP
 - Let E be an arbitrary instance of SAT
 - $C = (x_1 + x_2 + \dots + x_k)$ arbitrary clause of E
 - And $k \geq 4$
 - $C' = (x_1 + x_2 + y_1) \bullet (x_3 + \overline{y_1} + y_2) \bullet (x_4 + \overline{y_2} + y_3) \bullet \dots \bullet (x_{k-1} + x_k + y_{k-3})$

NP-Completeness Proof: 3SAT

- C' is satisfiable iff C is satisfiable:
- $C' = 1 \Rightarrow$ at least one $x_i = 1 \Rightarrow C = 1$
 - $C = 1 \Rightarrow$ at least one $x_i = 1 \Rightarrow$ can set y_i 's so that $C' = 1$

$$C = (x_1 + x_2)$$

$$C' = (x_1 + x_2 + y_1) \bullet (x_1 + x_2 + \overline{y_1})$$

$$C = (x_1)$$

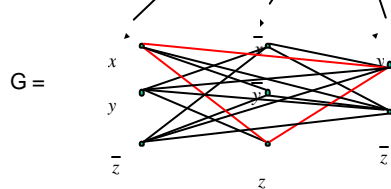
$$C' = (x_1 + y_1 + y_2) \bullet (x_1 + \overline{y_1} + y_2) \bullet (x_1 + y_1 + \overline{y_2}) \bullet (x_1 + \overline{y_1} + \overline{y_2})$$

NP-Completeness Proof: CLIQUE

- **Problem:** Does $G=(V,E)$ contain a clique of size k ?
- **Theorem:** Clique is NP-Complete. (reduction from SAT)
- **Idea:** Make "column" of k vertices for each clause with k variables.
 - No edge within a column.
 - All other edges present except between x and \overline{x}

NP-Completeness Proof: CLIQUE

- **Example:** $E = (x + \overline{y} + z) \bullet (\overline{x} + \overline{y} + z) \bullet (y + \overline{x})$



- G has m -clique (m is the number of clauses in E), iff E is satisfiable. (Assign value 1 to all variables in clique)