NP-Completeness

Preliminaries

- Polynomial-Time algorithm:

  On input size n the worst-case running time is $O(n^k)$ for some constant k.

- Turing's Halting problem:

  Not solvable by any computers

- Tractability: Synonymous to polynomial-time.

NP- Complete Problems
(an interesting class)

- **No polynomial time** algorithm has yet been discovered for an NP-Complete problem

- No one has been able to prove that **no polynomial time algo can exist for any one of them**

- $P \neq NP$ $\Longrightarrow$ most perplexing open research problem

Polynomial-time Solvable VS NP-Completeness

Polynomial-time Solvable

- Shortest Path
- Euler Tour
- 2-CNF

NP-Completeness

- Longest Simple Path (NP-Complete even if all edge weight are 1)
- Hamiltonian Cycle
- 3-CNF

Continued.

- P consists of problems solvable in polynomial time.
- NP consists problems that are verifiable in polynomial time
- If we were given a certificate of a solution , we can verify whether it is correct in polynomial time
- P $\subseteq$ NP ?
- (Open question whether P is a proper subset of NP)

NP-Complete Definition

- A problem P’ is in NP and it is as hard as any problem in NP
- If any NP-Complete problem has a polynomial time algorithm , then every one will have one.

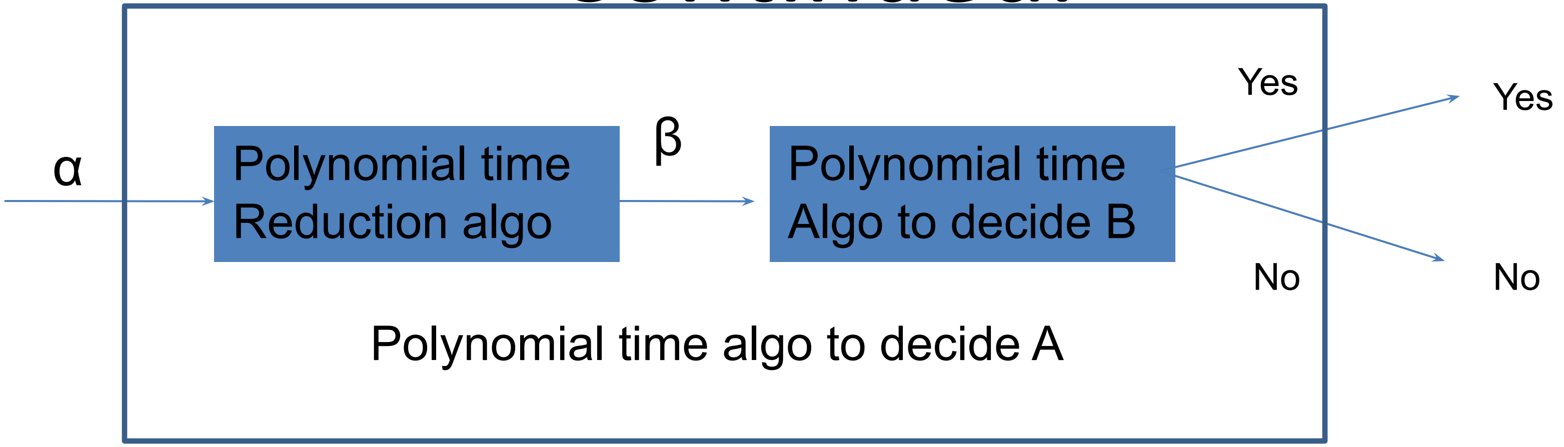
Optimization VS Decision Problem

- Given an instance α of problem A, use a polynomial time reduction algorithm to transform it to an instance β of B.

- Run the polynomial time decision algorithm for B on the instance β

- Use the answer for β as the answer for α

Continued.



- By reducing solving problem A to solving problem B, we use the easiness of ‘B’ to prove easiness of ‘A’.
- The same technique can now be used to show the same hardness of B if we already know the hardness of A.

Continued.

- We will prove that B is NP-Complete by using the result that A is NP-complete.

- First NP-Complete problem is required.

- 3-SAT was first proved to be NP-Complete.

Formal Language Framework

- An alphabet ∑ is a finite set of symbols
- A language L over ∑ is any set of strings made up of symbols from ∑.
- Example: ∑ = {0, 1}, L = {10, 11, 101, 111, 1011, ……….} is the language representing the binary strings for all prime numbers
- Union and Intersection are defined on language

Continued.

- Complement of L is $L' = \sum^{*} - L$

- Concatenation of 2 languages L_1 and L_2 is the language $L = \{X_1X_2: X_1 \in L_1 \text{ and } X_2 \in L_2\}$

- Closure or kleen star of a language L is the language $L^* = \{\in\} \cup L \cup L^2 \cup L^3 \cup \ldots\ldots.$

  where L^k is the language obtained by concatenating L to itself k times.

- The set of instances of any decision problemQ is simply the set $\sum^*$, where $\sum = \{0, 1\}$ since Q is entirely characterized by those problem instances that produce a 1 / (yes) answer, we can view Q as a language L over $\sum = \{0, 1\}$, where $L = \{x \in \sum^*: Q(x) = 1\}$
- We say an algo A accepts a string $x \in \{0, 1\}^*$, if given input x, the output of the algorithm A(x) is 1. $L = \{x \in \{0, 1\}^*: A(x) = 1\}$
- A language L is decided by an algorithm A if every binary string in L is accepted by A and every binary string not in L is rejected by A

A tentative definition of complexity Class P

- $P = \{L \subseteq \{0, 1\}^*: \exists$ an algo. A that decides L in polynomial time$\}$.

Verification Algorithm

- Two argument algorithm A, Where one argument is an ordinary input string x and the other is a binary string y called a certificate.

- A 2-argument algorithm A verifies an input string x if there exists a certificate y such that $A(x, y) = 1$.

Complexity class NP

- The class of languages that can be verified by a polynomial time algo.
- More precisely a language L belongs to NP if $\exists$ a two input poly-time algo A and a constant c such that

  L = {x $\in$ {0, 1}*: $\exists$ certificate y with $|y| = O(|x|^c)$ such that A(x, y) = 1.}
- $L_1 \leq_p L_2$ $\Rightarrow$ L_1 is not more than a polynomial factor harder than L_2

NP Complete language

- A language L $\subseteq$ {0, 1}* is NP-complete if
  - ❑ L $\in$ NP, and
  - ❑ L' $\leq_p$ L for every L' $\in$ NP
  - ❑ If only condition 2 is satisfied then L is NP-hard

Optimization Problem

- Find the maximum sized clique from a graph

- The corresponding decision problem called clique decision problem (CDP) is NP-Complete

- CNF satisfiability is NP- Complete by Cook-Levin theorem.

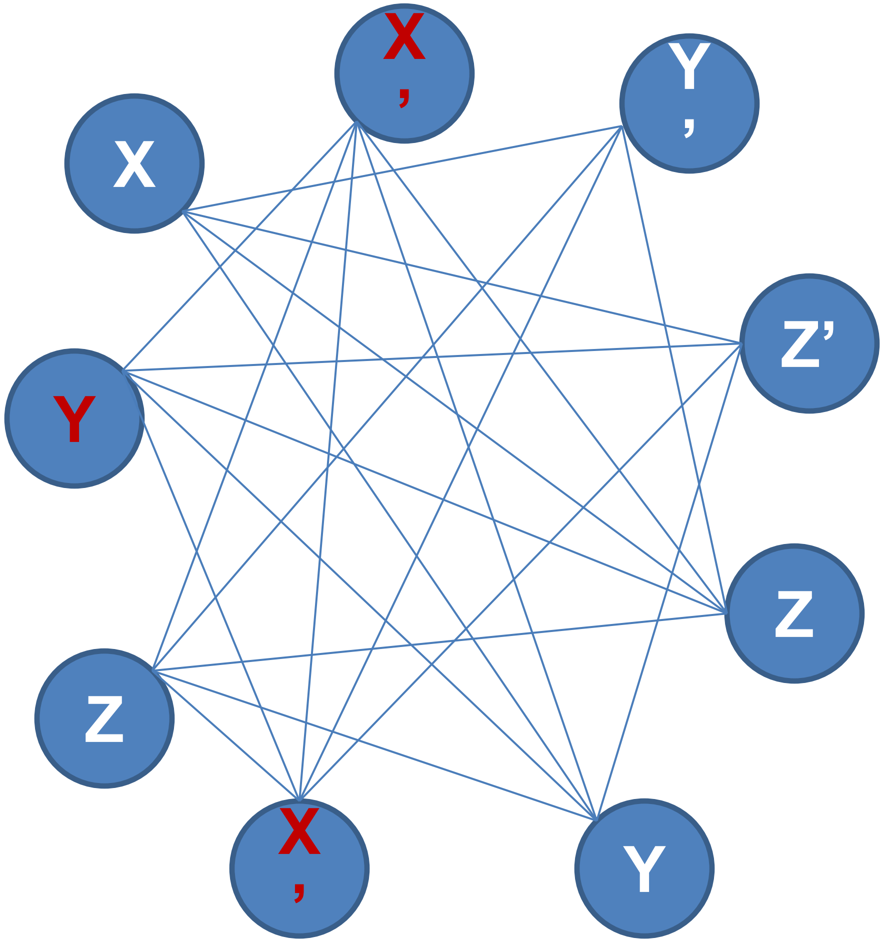
- Karp reduced CNF-Sat to CDP.

Construction

- CNF formula(f) → Graph (G)
- It has a vertex for every pair (v, c).
  - v is a variable or its negation (v’),
  - c is a clause in the formula f that contains v.
- Edges are there between (v, c) and (u, d) if , c ≠ d and u ≠ v’
- Means: between any 2 literals in different clauses, who are not each other’s negation.

Example

(x U y U z) ^ (x’ U y’ U z’) ^ (x’ U y U z)



The 3 highlighted vertices give a 3-clique and correspond to a satisfying assignment.

Continued.

- If k denotes the no. of clauses in the CNF formula (f), then the k-vertex clique in the graph represents ways of assigning truth values to some of its variables in order to satisfy the formula.

- f is satisfiable iff a k-clique exists.