CSE 589
Applied Algorithms
Spring 1999

NP-Completeness

NP-Completeness Theory

- Explains why some problems are hard and probably not solvable in polynomial time.
- Invented by Cook in 1971.
- Popularized in an important paper by Karp in 1972.
- Standardized by Garey and Johnson in 1979 in “Computers and Intractability: A Guide to the Theory of NP-Completeness”.

NP

- NP stands for nondeterministic polynomial time.
- We consider the class of decision problems (yes/no problems).
- A nondeterministic algorithm is one that can make “guesses”.
- A decision problem is in NP if it can be solved by a nondeterministic algorithm that runs in polynomial time.

Examples of Decision Problems in NP

- Hamiltonian Path
  - Nondeterministic algorithm: guess a path $v_1, v_2, \ldots v_n$ then check no two vertices are the same and that for each i < n there is an edge between v_i and v_{i+1}.
- Graph Coloring
  - input: Graph G = (V,E) and a number k.
  - output: Determine if all vertices can be colored with k colors such that no two adjacent vertices have the same color.
  - Algorithm: Guess a coloring and then check it.

CNF-Satisfiability

- Input: A Boolean formula F in conjunctive normal form.

$$(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

- Output: Determine if F is satisfiable, that is, there is some assignment to the variables that makes the formula F true.

$$x = 1,\ y = 0,\ z = 1$$

$$(1 \vee 0 \vee 1) \wedge (\neg 1 \vee 0 \vee 1) \wedge (\neg 1 \vee \neg 0 \vee \neg 1)$$

- Algorithm: Guess an assignment and check it.

Subset Sum

- Input: Integers $a_1, a_2, \ldots, a_n, b$
- Output: Determine if there is subset

$$X \subseteq \{1,2,\ldots,n\}$$

with the property $\sum_{i \in X} a_i = b$

- Algorithm: Guess the subset X and check the sum adds up to b.

Decision Problems are Polynomial Time Equivalent to their Reporting Problems

- Example: Subset sum
  - Decision Problem: Determine if a subset sum exists.
  - Reporting Problem: Determine if a subset sum exists and report one if it does.
- Using decision to report
  - Let subset-sum(A,b) return true if some subset of A adds up to b. Otherwise it returns false.

Reporting Reduces to Decision

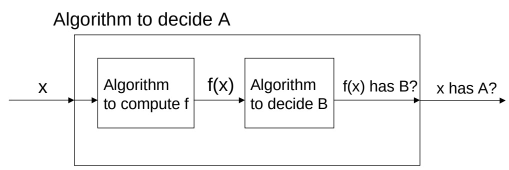
```
Assume that subset-sum ({a_1,…,a_n},b) is true
X := the empty set;
for i = 1 to n do
   if subset-sum({a_{i+1},…,a_n},b - a_i) then
      add i to X;
      b := b - a_i;
```

Example: 3, 5, 2, 7, 4, 2, b = 11
5, 2, 7, 4, 2, b = 11-3 --> yes, X = {3}, b = 8
2, 7, 4, 2, b = 8-5 --> no
7, 4, 2, b = 8-2 --> yes, X = {3,2}, b = 6
4, 2, b = 6-7 --> no
2, b = 6-4 --> yes, X = {3,2,4}, b = 2
b = 4 -2 --> yes, X ={3,2,4,2}

Polynomial Time Reducibility

- Informal idea: A decision problem *A* is polynomial time reducible to a decision problem *B* if a polynomial time algorithm for *B* can be used to construct a polynomial time algorithm for *A*.
- Formally: *A* is polynomial time reducible to *B* if there is a function *f* computable in polynomial time such that for all *x*:
  - *x* has *A* if and only if *f(x)* has *B*
- If *A* polynomial time reducible to *B* and *B* solvable in polynomial time then so is *A*.

Block Diagram to Decide A from B



Example of Polynomial Time Reduction

- Hamiltonian path is polynomial time reducible to spanning tree of degree 4.
  - Given *G = (V,E)*
  - Construct *G' = (V',E')*
  - *f(G) = G'*
  - *G* has Hamiltonian path if and only if *G'* has a spanning tree of degree 4

NP-Hardness

- Definition: A problem *A* is NP-hard if every problem in NP is reducible to it in polynomial time.
- If an NP-hard problem has a polynomial time algorithm, then every problem in NP has a polynomial time algorithm.
- To show a problem is NP-hard it suffices to show that some NP-hard problem is reducible to it. Why? Transitivity of polynomial time reduction.

Transitivity of Polynomial Time Reduction

- Define: $A \le_P B$ to mean that A is polynomial time reducible to B.
- Transitivity: $A \le_P B$ and $B \le_P C$ implies $A \le_P C$
- Example:
  - Every problem in NP is known to be polynomial time reducible to Hamiltonian path.
  - Hamiltonian path is polynomial time reducible to spanning tree of degree 4.
  - Therefore, every problem in NP is polynomial time reducible to spanning tree of degree 4.

NP-Completeness Definition

- Definition: A decision problem A is NP-complete if
  - A is in NP
  - A is NP-hard
- Example: Spanning tree of degree 4 is NP-complete.
  - Spanning tree of degree 4 is in NP.
  - Hamiltonian path is a known NP-complete problem.
  - Hamiltonian path is polynomial time reducible to spanning tree of degree 4.

Cook's Theorem

- CNF-satisfiability is NP-complete
  - Cook 1971

Proof formalizes the notion of a nondeterministic algorithm as a nondeterministic Turing machine. Cook then shows that a CNF-formula F can be produced in polynomial time that describes the operation of the nondeterministic Turning machine. The Turing machine halts in a "yes" state if and only if the formula F is satisfiable.

P vs NP

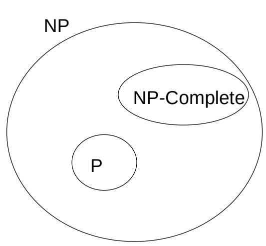
- Definition: P is the class of decision problems that are solvable by a polynomial time algorithm.
- Every problem in P is also in NP

$$P \subseteq NP$$

- Famous Open Question:

$$P = NP?$$

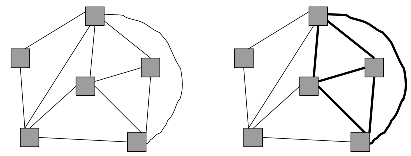
Probable Picture



Clique Decision Problem

- Input: Undirected Graph G = (V,E) and a number k.
- Output: Determine if G has a k-clique, that is, there is a set of vertices U of size k such that for each pair of vertices in U there is and edge in E between them.

Clique Example



Clique is NP-Complete

- Clique is in NP
  - Nondeterministic algorithm: guess k vertices then check that there is an edge between each pair of them.
- Clique is NP-hard
  - We reduce CNF-satisfiability to Clique in polynomial time
  - Given a CNF formula F we need to construct a graph G and a number k with the property that F is satisfiable if and only if G has a k-clique. The contstruction must be efficient, polynomial time.

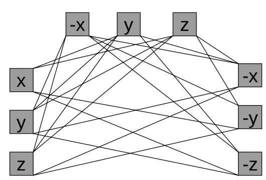
Construction by Example

$$F = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge \underline{(\neg x \vee \neg y \vee \neg z)}$$

literal

clause

G

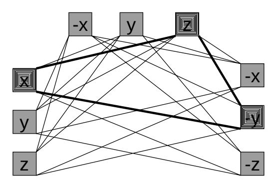


Construction by Example

$$F = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

$$x = 1,\ y = 0,\ z = 1$$

G



General Construction

$F = \bigcap_{i=1}^{k} \bigcup_{j=1}^{m_i} a_{ij}$ where $a_{ij} \in \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$

literals

$G = (V, E)$ where

$$V = \{a_{ij} : 1 \le i \le k, 1 \le j \le m_i\}$$

$$E = \{\{a_{ij}, a_{i'j'}\} : i \ne i' \textit{and},$$
$$a_{ij} \textit{ and } a_{i'j'} \textit{ are not complementary}\}$$

$$k \textit{ is the number of clauses}$$

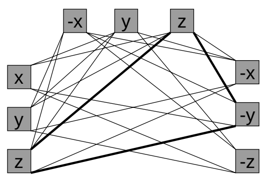
The Reduction Argument

- We must show
  - F satisfiable implies G has a clique of size k.
    - Given a satisfying assignment for F, for each clause pick a literal that is satisfied. Those literals in the graph G form a k-clique.
  - G has a clique of size k implies F is satisfiable.
    - Given a k-clique in G, assign each literal in the clique to be 1. This yields a satisfying assignment to F.

Clique to Assignment

$$F = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

G

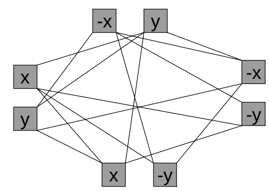


$$y = 0,\ z = 1$$

Assignment to Clique

$$F = (x \vee y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y) \wedge (x \vee \neg y)$$

G



G has no 4-clique

3-CNF-Satifiability

- Input: A Boolean formula F with at most 3 literals per clause.
- Output: Determine if F is satisfiable.

- 3-CNF-Satisfiability is NP-complete
  - This is probably the most used NP-complete problem in reduction proofs showing other decision problems are NP-hard.

Reduction by Example

Given $F = (x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge F'$

Construct $H = (x_1 \vee z_1) \wedge (\neg x_2 \vee \neg z_1 \vee z_2)$
$\wedge (x_3 \vee \neg z_2 \vee z_3) \wedge (\neg x_4 \vee \neg z_3) \wedge F'$

F is satisfiable if and only if *H* is satisfiable.

$x_2 = 0$ satisfies the first clause of *F*.

$z_1 = 1,\ z_2 = 0,\ z_3 = 0$ satisfy clauses 1,3, and 4 of *H* and

$x_2 = 0$ satisfies the clause 2 of *H*.