NP Hard and NP-Complete Problems

- Problems (Ptime tractable):
  1. Sorting
  2. Searching
  3. Max Flow
  4. Matching
  5. Shortest Path
- NP-Complete Problems:
  1. SAT
  2. Clique
  3. Node Cover
  4. Hamiltonian Cycle
- Exponential Time Problems (intractable):
  1. Enumeration Problems
- Undecidable Problems:
  1. Halting Problem
  2. Post Correspondence Problem

Definitions

- Optimization vs Decision
- Deterministic vs Non-Deterministic
  - Deterministic - outcome of every operation is uniquely defined
  - Non-deterministic - outcomes of some operations are not uniquely defined but are limited to specific sets of possibilities

Non-Deterministic Algorithms

- To specify non-deterministic algorithms, we introduce three new functions:
  1. Choice(s) – arbitrarily chooses one of the elements of set .
  2. Failure() – signals an unsuccessful completion.
  3. Success() – Signals a successful completion.
- Whenever there is a set of choices that leads to a successful completion, then one such set of choices is always made and the algorithm terminates successfully
- A non-deterministic algorithm terminates unsuccessfully iff there exists no set of choices to a success signal
- Choice(), Success(), Failure() takes $O(1)$ time

Non-Deterministic Search Algorithm

```
ND-Search Algorithm
  j:=Choice(1,n)
  if A(j) = x then
  {
    write(j);
    success();
  }
  write (0); Failure();
```

$O(1)$ time Any deterministic algorithm must take $\Omega(n)$.

```
Algorithm NSort (A, n)
//Sort n positive integers
//using a non-deterministic algorithm.
{
   for i := 1 to n do B[i] := 0;
   for i := 1 to n do
   {
      j:= Choice(1, n);
      if B[j] ≠ 0 then Failure();
      B[j] := A[i];
   }
   for i := 1 to n − 1 do
      if B[i] > B[i + 1] then Failure();
   write(B[1 : n]);
   success();
}
```


NP(Non-Deterministic Polynomial) Problems

- Maximum Clique
- 0/1 knapsack
- Satisfiability
- Node cover
- Hamiltonian Cycle
- Traveling Salesman Problem (TSP)

Maximum Clique

```
Algorithm NClique (g, n, k)
{
   S := ∅; // S is an empty set
   for i := 1 to k do
   {
      t:= Choice(1, n);
      if t ∈ S then Failure();
      S := S ⋃ {t}; //Add t to S
   }
   for all pairs (i, j) such that i ∈ S, j ∈ S,
   and i ≠ j do
      if (i, j) is not an edge of g then Failure();

   Success();
}
```


Satisfiability

```
Algorithm NSAT (E, n)
//Determine whether the proposition formula
//E is satifiable. The variables are x_1, ..., x_n
{
   for i := 1 to n do
      x_i:= Choice(false,true);
   if E(x_1, ..., x_n) then Success();
   else Failure();
}
```

P vs NP

P is the set of all decision problems solvable by deterministic algorithms in polynomial time.

NP is the set of all decision problems solvable by non-deterministic algorithms in polynomial time.

Clearly,

$P \subseteq NP$
$P = NP$?
$P \neq NP$?
$NP \subseteq P$?

This is one of the most important questions in Computer Science and Mathematics today.

In fact, there is a prize money of one million dollars if you can solve the $NP = P$ problem.

Cook's Theorem

Theorem: Satisfiability is in P iff P=NP.

- It is a landmark theorem.

- It says that all NP problems $\propto$ (reducible) satifiability

- How can you prove that **all** NP-Problems are reducible?

- It is done by showing that every NP-algorithm using the NP Turing machine can be expressed as boolean equation such that the equation returns true iff the NP-algorithm returns success and returns false if the NP-algorithms returns failure.

$\propto_p$: Reducibility in Polynomial Time

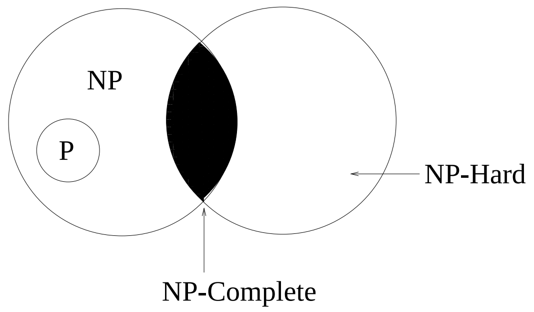
Let L_1 and L_2 be two problems.

Problem L_1 reduces ($\propto_p$) to L_2 iff there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time.

NP-Hardness and NP-Completeness

A problem L is NP-hard iff satisfiability reduces to L (SAT $\propto L$).

A Problem L is NP-complete iff L is NP-hard and $L \in NP$.



"Commonly believed relationship"

Problem Relationship

- 2 problems L_1 and L_2 are said to be polynomially equivalent iff $L_1 \propto L_2$ and $L_2 \propto L_1$
- To show that a problem L_2 is NP-hard, it is adequate to show $L_1 \propto L_2$ where L_1 is some problem already known to be NP-hard. Given $\propto$ is a transition relation, it follows that if $SAT \propto L_1$ and $L_1 \propto L_2$ then $SAT \propto L_2$
- For L_2 to be NP-complete L_2 must be in NP

Theorems

Cook's Theorem : Please refer to the CLR Intro to algorithms book.

CNF : We assume that SAT $\propto$ CNF. It is trivial to show that CNF is in NP, hence CNF is NP-Complete.

Theorem : (CNF $\propto$ CDP) CNF-Satisfiability $\propto$ Clique Decision Problem.

Theorem : (CDP $\propto$ NCP) Clique Decision Problem $\propto$ Node Cover Problem.

Theorem : (CNF $\propto$ 3SAT) CNF $\propto$ 3SAT.

Others : Check the book "Computers and Intractability" by Garey and Johnson or "A Compendium of NP optimization problems" by Crescenzi and Kann.

Theorem : CNF-Satisfiability $\propto$ Clique Decision Problem

Proof:

Let $F = \wedge_{1 \leq i \leq k} C_i$ be a propositional formula in CNF. Let x_i, $1 \leq i \leq n$, be variables in F.

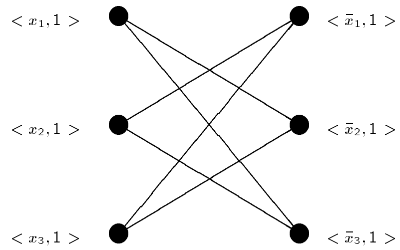
We construct a graph $G(V, E)$ such that G has a clique of size at least k iff F is satisfiable.

If the size of F is m, G should be obtainable from F in $O(p(m))$ time. Hence, if we have a ptime algorithm for CDP then we can obtain a ptime algorithm for CNF.

For any F, $G(V, E)$ is defined as follows:
$V = \{ <\sigma, i> | \sigma$ is a literal in Clause C_i $\}$
$E = \{ (<\sigma, i>, <\delta, j>) | i \neq j$ and $\sigma \neq \bar{\delta} \}$

An Example

$F = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$



Claim: F is satisfiable iff G has a clique of size $\geq K$.

Proof of claim:

$\Rightarrow$
If F is satisfiable, then there is a set of true values for x_i, $1 \leq i \leq n$, such that each clause is true with this assignment.

Thus, there is at least one literal σ in each C_i such that σ is true.

Let $S = \{ <\sigma, i> | \sigma$ is true in $C_i\}$ be a set containing exactly one $<\sigma, i>$ and $<\delta, j>$ in S, there is an edge in G, since $i \neq j$ and both σ and δ have the value true.

Thus, S forms a clique in G of size K.

$\Leftarrow$
If G has a clique $K = (V', E')$ of size at least K, let $S = \{<\sigma, i> | <\sigma, i> \in V' \}$. Clearly $|S| = K$ as G has no clique more than K.

If $S' = \{\sigma | <\sigma, i> \in S$ for some i $\}$, then S' cannot contain both a literal δ and its complement $\bar{\delta}$ as there is no edge connecting both $<\delta, i>$ and $<\bar{\delta}, j>$ in G. Hence by setting x_i=true if $x_i \in S'$ and x_i=false if $\bar{x}_i \in S'$ and choosing arbitrary true values for variables not in S', we can satisfy all clauses in F. Hence, F is satisfiable iff G has a clique of size at least K.

CDP $\propto$ NCP

Thereom: CDP $\propto$ NCP

Proof:

Let $G(V, E)$ and K define an instance of CDP. Assume $|V| = n$. We construct a graph G'. G' has a node cover of size at most $n - k$ iff G has a clique of at least K.

Graph $G' = (V, \bar{E})$.

If G has a clique size of at least K, then G' has a node cover of at most $n - k$.

Let K be any clique in G. Since there are no edges in $\bar{E}$ converting vertices in K, the remaining $n - |k|$ vertices in G' must cover all edges in $\bar{E}$. Similarly, if S is a node cover of G', then $V - S$ must form a complete subgraph in G.

CNF $\propto$ 3SAT

Theorem: CNF $\propto$ 3SAT

CNF		3SAT
$(x_1, x_2, \bar{x}_3)$	$\longrightarrow$	$(x_1, x_2, \bar{x}_3)$
(x_1, x_2)	$\longrightarrow$	(x_1, x_2, y_1)
		$(x_1, x_2, \bar{y}_1)$
(x_1)	$\longrightarrow$	(x_1, y_1, y_2)
		$(x_1, \bar{y}_1, y_2)$
		$(x_1, \bar{y}_2, y_1)$
		$(x_1, \bar{y}_1, \bar{y}_2)$
$(x_1, x_2, ..., x_k)$	$\longrightarrow$	(x_1, x_2, y_1)
		$(\bar{y}_1, x_3, ..., x_k)$
	$\longrightarrow$	(x_1, x_2, y_1)
		$(\bar{y}_1, x_3, y_2)$
		$(\bar{y}_2, x_4, ..., x_k)$
		.
		.
		.