1. The class of complement problems \(\text{coNP}\)

- \(\text{NP}\) is the class of problems with succinct certificates.
- \(\text{coNP}\) is the class of problems with succinct disqualifications.

**Example.** Consider the problem of VALIDITY:

INSTANCE: A Boolean expression \(\phi\) in CNF.

QUESTION: Is \(\phi\) valid?

- VALIDITY is in \(\text{coNP}\): for an expression \(\phi\) which is not valid, a falsifying truth assignment is a succinct disqualification.
- \(\text{HAMILTON PATH COMPLEMENT}\) and \(\text{SAT COMPLEMENT}\) are also in \(\text{coNP}\).
- \(\text{P} \subseteq \text{coNP}\)

2. The Relationship of \(\text{coNP}\) and \(\text{NP}\)

**Proposition.** If \(L \subseteq \Sigma^*\) is \(\text{NP}\)-complete, then its complement \(L' = \Sigma^* - L\) is \(\text{coNP}\)-complete.

Further observations:

- It is open whether \(\text{NP} = \text{coNP}\).
- If \(\text{P} = \text{NP}\), then \(\text{NP} = \text{coNP}\) (and \(\text{P} = \text{coNP}\)).
- It is possible that \(\text{P} \neq \text{NP}\) but \(\text{NP} = \text{coNP}\) (however, it is strongly believed that \(\text{NP} \neq \text{coNP}\)).
- The problems in \(\text{coNP}\) that are \(\text{coNP}\)-complete are the least likely problems to be in \(\text{P}\) and also in \(\text{NP}\) (see below).
Do \( \text{coNP} \) and \( \text{NP} \) coincide?

**Proposition.** If a \( \text{coNP} \)-complete problem is in \( \text{NP} \), \( \text{NP} = \text{coNP} \).

**Proof.** Suppose that \( L \) is a \( \text{coNP} \)-complete problem that is in \( \text{NP} \).

(\( \supseteq \)) Consider \( L' \in \text{coNP} \). Then there is a reduction \( R \) from \( L' \) to \( L \). Then \( L' \in \text{NP} \), because \( L' \) can be decided by a polynomial time NTM which on input \( x \) computes first \( R(x) \) and then starts the NTM for \( L \).

(\( \subseteq \)) Consider \( L' \in \text{NP} \). Then \( L' \in \text{coNP} \) and there is a reduction \( R \) from \( L' \) to \( L \). Then similarly \( L' \in \text{NP} \) and hence \( L' \in \text{coNP} \). \( \square \)

The primality problem \( \text{PRIMES} \)

**INSTANCE:** An integer \( N \) in binary representation.

**QUESTION:** Is \( N \) a prime number?

- \( \text{PRIMES} \in \text{coNP} \) as any divisor acts as a succinct disqualification.
- Note that a \( O(\sqrt{N}) \) algorithm for \( \text{PRIMES} \) testing all relevant divisor candidates is only pseudopolynomial.
- \( \text{PRIMES} \in \text{NP} \) (as shown below) and hence \( \text{PRIMES} \in \text{coNP} \cap \text{NP} \).
- New result in August 2002: M. Agrawal, N. Kayal, N. Saxena: \( \text{PRIMES} \) is in \( \text{P} \) !!

**PRIMES has succinct certificates**

A succinct certificate for primality can be obtained using the following theorem.

**Theorem.** A number \( p > 1 \) is prime iff there is a number \( 1 < r < p \) such that \( r^{p-1} = 1 \mod p \) and, furthermore, \( r^{q-1} \neq 1 \mod p \) for all prime divisors \( q \) of \( p - 1 \).

**Corollary.** \( \text{PRIMES} \) is in \( \text{NP} \cap \text{coNP} \).

- The theorem provides a succinct certificate for the primality of \( p \):
  \[ C(p) = (r;q_1,C(q_1),\ldots,q_k,C(q_k)) \]
  where \( C(q_i) \) is a recursive primality certificate for each prime divisor \( q_i \) of \( p - 1 \).
- The recursion stops for prime divisors \( q_i = 2 \) for which \( C(q_i) = (1) \).
Verifying the certificate $C(p)$

The following observations can be made:

➤ The certificate $C(p)$ is polynomial in the length of $p$ (in $\log p$) and it can be checked by division and exponentiation.

➤ Ordinary multiplication and division are doable in polynomial time in the length of the input (in binary representation).

➤ Exponentiation $r^{p-1} \mod p$ can be done in polynomial time by repeated squaring $r^1, r^2, r^4, \ldots, r^{2^l} \mod p$ where $l = \lfloor \log_2(p-1) \rfloor$ and then with at most $l$ additional multiplications.

The certificate $C(p)$ can be checked in polynomial time.

4. Function Problems vs. Decision Problems

➤ We have studied decision problems but many problems in practice require a more complicated answer than “yes” / “no”.

Example. Find a satisfying truth assignment for a formula.

Example. Compute an optimal tour for TSP.

➤ Such problems are called function problems.

➤ Decision problems are useful surrogates of function problems only in the context of negative complexity results.

Example. SAT and TSP(D) are NP-complete. Then unless $P = NP$, there is no polynomial time algorithm for finding a satisfying truth assignment or an optimal tour.

The relationship of SAT and FSAT

FSAT: given a Boolean expression $\phi$, if $\phi$ is satisfiable then return a satisfying truth assignment of $\phi$ otherwise return “no”.

➤ If FSAT can solved in polynomial time, then clearly so can SAT.

➤ If SAT can be solved in polynomial time, then so can FSAT using the following algorithm given input $\phi$ with variables $x_1, \ldots, x_n$ ($\phi[x = \text{true}]$ denotes $\phi$ where variable $x$ is replaced by true):

1. if $\phi \notin \text{SAT}$ then return “no”;
2. for all $x \in \{x_1, \ldots, x_n\}$ do
   a. if $\phi[x = \text{true}] \in \text{SAT}$ then $T(x) := \text{true}$; $\phi := \phi[x = \text{true}]$
   b. else $T(x) := \text{false}$; $\phi := \phi[x = \text{false}]$
3. return $T$;

The relationship of TSP(D) and TSP

➤ If TSP can solved in polynomial time, then clearly so can TSP(D).

➤ If TSP(D) can solved in polynomial time, then so can TSP in the following way.

1. An optimal tour can be found using the algorithm below which finds
   a. the cost $0 \leq C \leq 2^n$ of an optimal tour by binary search and
   b. an optimal tour using the cost $C$ computed in step 1.

2. (Here $n$ is the length of the encoding of the problem instance.)

3. Both steps involve a polynomial number of calls to the polynomial time algorithm for TSP(D) (given such an algorithm exists).
An algorithm for TSP

An algorithm for TSP(D) is used as a subroutine:

```c
/* Find the cost C of an optimal tour by binary search*/
C := 0; C_u := 2^n;
while (C_u > C) do
  if there is a tour of cost \lfloor (C_u + C)/2 \rfloor or less then
    C_u := \lfloor (C_u + C)/2 \rfloor;
  else
    C := \lfloor (C_u + C)/2 \rfloor + 1;
  /* Find an optimal tour */
  For all intercity distances do
    set the distance to C_u + 1;
    if there is a tour of cost C or less, freeze the distance to C_u + 1
    else restore the original distance and add it to the tour;
endfor
```

5. Classes of Function Problems

Definition. Let \( L \in \text{NP} \). Then there is a polynomial time decidable and polynomially balanced relation \( R_L \) such that for all strings \( x \), there is a string \( y \) with \( R_L(x, y) \) iff \( x \in L \).

The function problem associated with \( L \) (denoted \( F_L \)) is:

- The class of all function problems associated as above with languages in \( \text{NP} \) is called \( \text{FNP} \).
- \( \text{FP} \) is the subclass of \( \text{FNP} \) solvable in polynomial time.
- FSAT is in \( \text{FNP} \) and FHORNSAT is in \( \text{FP} \) (but it is open whether TSP is in \( \text{FNP} \)).

Reductions and completeness for function problems

A function problem \( A \) reduces to a function problem \( B \) if there are string functions \( R, S \) computable in logarithmic space such that for all strings \( x, z \): if \( x \) is an instance of \( A \), then \( R(x) \) is an instance of \( B \) and if \( z \) is a correct output of \( R(x) \), then \( S(z) \) is a correct output of \( x \).

- Reductions compose among function problems.
- A problem \( A \) is complete for a class \( \mathcal{F}_C \) of function problems if it is in \( \mathcal{F}_C \) and every problem in \( \mathcal{F}_C \) reduces to \( A \).
- \( \text{FP} \) and \( \text{FNP} \) are closed under reductions.
- FSAT is \( \text{FNP} \)-complete.
- \( \text{FP} = \text{FNP} \) iff \( P = \text{NP} \).

6. Total Functions

- There are certain important problems in \( \text{FNP} \) that are guaranteed to never return "no".

Example. FACTORING: Given an integer \( N \), find its prime decomposition \( N = p_1^{k_1} \cdots p_m^{k_m} \).
(No known polynomial time algorithm).

- FACTORING seems to be different from the other hard problems in \( \text{FNP} \); it is a total function in a sense:

Definition. A problem \( L \) in \( \text{FNP} \) is called total if for every string \( x \) there is at least one string \( y \) such that \( R_L(x, y) \).

- The subclass of \( \text{FNP} \) containing all total function problems is denoted by \( \text{TFNP} \).
There are also other problems in TFNP with no known polynomial time algorithm.

**Example.** HAPPYNET:

**INSTANCE:** An undirected graph \( G = (V, E) \) with integer weights \( w \) on edges.

**GOAL:** Find a state of the graph where all nodes are happy.

- A state is a mapping \( S: V \mapsto \{-1, +1\} \).
- A node \( i \) is happy in a state \( S \) of \( G = (V, E) \) if
  \[ S(i) \cdot \sum_{[i, j] \in E} S(j)w[i, j] \geq 0. \]

**Properties of HAPPYNET**

- Every instance is guaranteed to have a happy state which can be found using the following algorithm:
  Start with any \( S \) and while there is an unhappy node, flip it.
- This algorithm is not polynomial but pseudopolynomial \( O(W) \) where \( W \) is the sum of all weights.
- No polynomial algorithm known.
- HAPPYNET equivalent with finding stable states in neural networks in the Hopfield model.

**Learning Objectives**

- The definition of coNP and examples of languages from this class, e.g., VALIDITY.
- The characterization of coNP based on disqualifications.
- Reductions and completeness for function problems
- Relationship of decision problems and function problems