NP Completeness

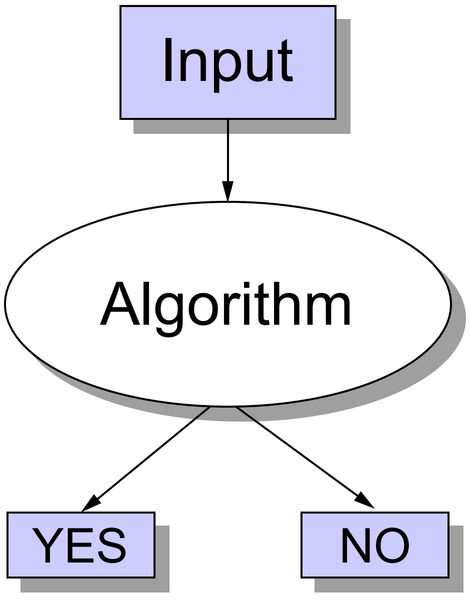
Yong Wen Chua

https://github.com/lawliet89/np-complete

Content

1. Decision Problems
2. What is the Computational complexity theory?
3. Turing Machine model of computation
4. Complexity Hierarchy
5. P
6. NP
7. Reduction
8. NP-Complete
9. P = NP?

Decision Problems



Reachability

`RCH`: Given a directed graph `G` and nodes `x`, `y`, is there a path from `x` to `y`?

RCH Algorithm

Let `S` be a set of nodes to process. There are `n` nodes in the graph.
Initially, $S = \{x\}$.

At each stage, for some $z \in S$:

- remove `z` from `S`
- mark `z`
- Find all unmarked "successors" of `z` and add them to `S`

Until `y` is found or `S` is empty.

Worst case: each edge is examined once. There are at most n^2 edges.
So $O(n^2)$.

(Formal) Big Oh Notation

Let $f, g : \mathbb{N} \to \mathbb{N}$.

Definition

$f(n) = O(g(n))$ if $\exists c, n_0 \in \mathbb{Z}_+$ such that $\forall n \geq n_0$ $f(n) \leq c.g(n)$.

The time or space required is bounded by this function.

Computation Complexity Theory

- More than Big Oh Notation
- Classifying computational problems according to their inherent difficulty into different *classes*
- Relation between the different *classes*

Decidable vs Undecidable

`RCH`: Decidable

`HALT`: Given the description of an *arbitrary* program and a finite input, decide whether the program finishes running or will run forever.

This is undecidable *over Turing Machines*.

Proceedings of the London Mathematical Society, Volume s2-42, Issue 1, 1 January 1937, Pages 230â€“265, https://doi.org/10.1112/plms/s2-42.1.230

`HALT`

Consider an *Oracle* $H(p, x)$ which decides that if some program p will halt for some input x. (i.e. a black box that solves `HALT`)

Then, we construct program `P`:

```
program P(y):
  if H(y,y) = halt then
    loop forever
  else:
    return
```

- If $H(P, P)$ = `halt` then $P(P)$ runs forever.
- If $H(P, P)$ = `loop` then $P(P)$ halts.

H always gives the wrong answer. Generalized H cannot exst.

âˆŽ Contradiction

Tractable vs Intractable

- Tractable problems can be solved in a *feasible* or *practical* amount of time

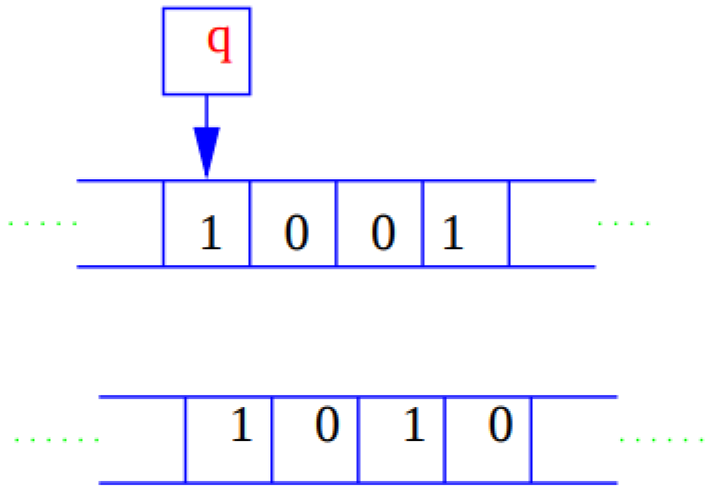
Cook-Karp Thesis: Tractable = polynomial time (P)

but... is n^{100} or $2^{n/100}$ more practical?

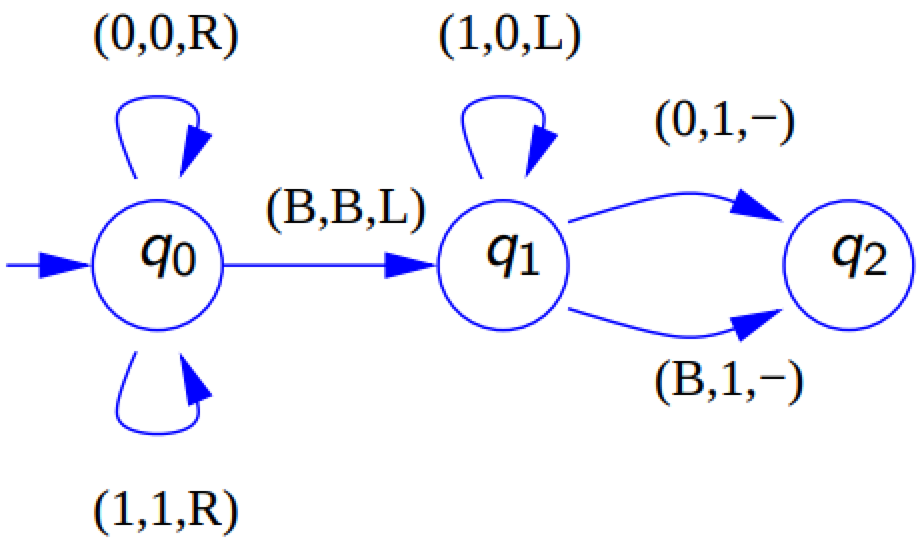
(Deterministic) Turing Machine

- Read/Write head over a tape. Can move right or left. The head stores the current *state* of the machine
- Symbols on tape
- A table of instructions where given the current state, and the symbol on the tape, decide whether to write to the tape, move left/right, or transition into a new state.

Turing Machine - Binary Successor



State Machine:



Universal Turing Machine (UTM)

- A Turing Machine that can take in an arbitrary program as input and run that program.
- "Stored-program computer"
- The basis of modern computing
- A multi-tape UTM is only slower by a logarithmic factor compared to the machine it simulates.

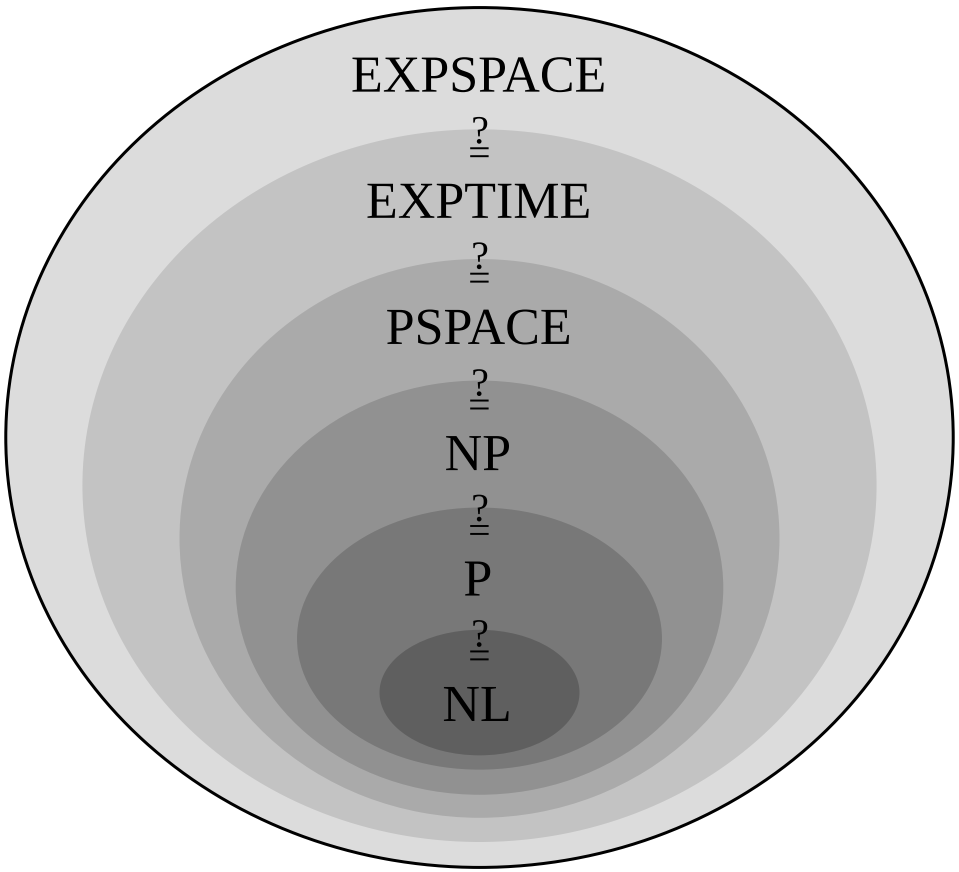
Invariance Thesis

(similar to Church-Turing Thesis)

All *reasonable* sequential models of computation have the same time complexity as Deterministic Turing Machines (DTM) up to a polynomial.

e.g. RAMs, 1-tape DTM, k-tape DTM

Complexity Hierarchy



(Formal) Languages

Formally, a language L is a set of strings over a given alphabet Σ.

Then let $L \subseteq (\Sigma - \{B\})^*$ and M be a DTM with alphabet Σ such that for any $w \in (\Sigma - \{B\})^*$:

- if $w \in L$ then $M(w)$ terminates with `yes`
- if $w \notin L$ then $M(w)$ terminates with `no`

Then we can say M **decides** L and L is recursive because it is decided by some DTM.

Then iff for any length $n = |w|$ of $w \in (\Sigma - \{B\})^*$, M operates within the time bound $f(n)$, we say

$$L \in TIME(f(n))$$

Polynomial Time (P)

$$P = \bigcup_k TIME(n^k)$$

That is P is the set of decision problems which can be decided by a DTM in polynomial time for all inputs.

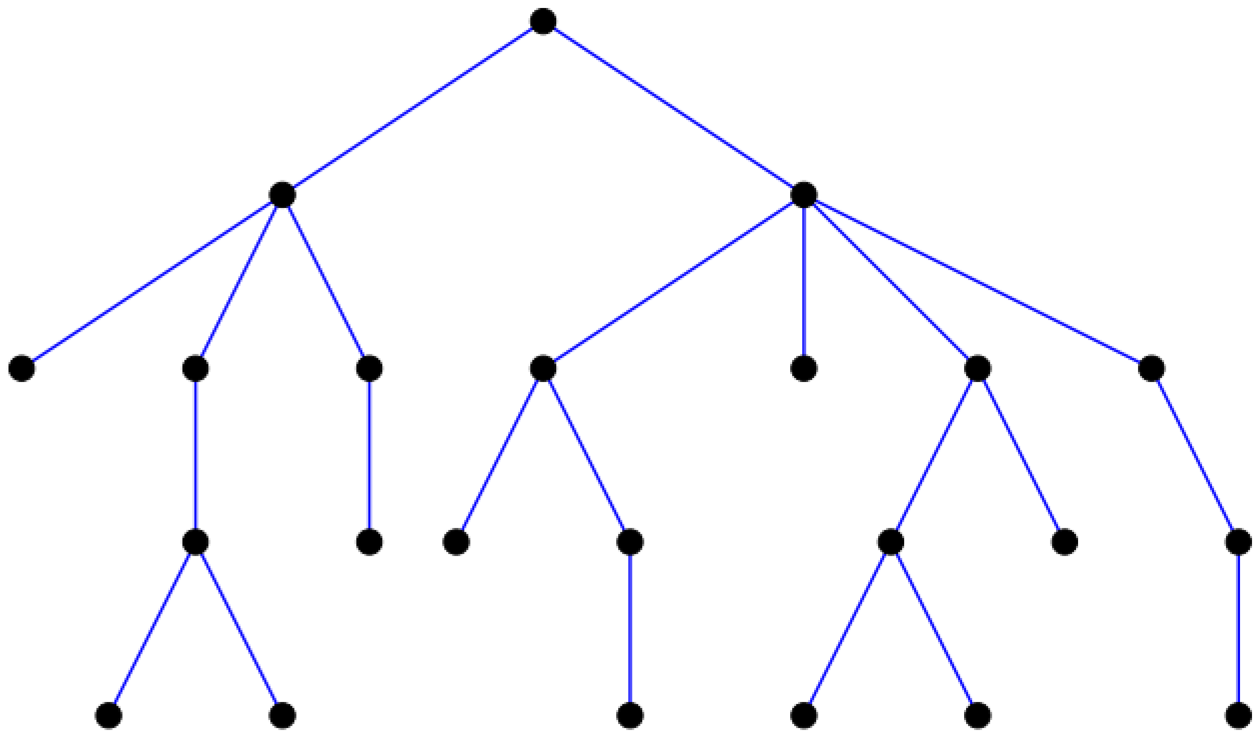
The function analogue to P is FP.

Known Problems in P

- Greatest Common Denominator (GCD)
- PRIMES - whether a number is prime (https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf)

Non-deterministic Turing Machines (NDTM)

A NDTM M has many possible computation for some input w that is chosen at random.



$M(w)$ represents the tree, with some depth.

NDTM is not practical.

(Formal) Nondeterministic Acceptance and Time

For some language L, if M returns `yes` on some input w, then we say M accetps L. (i.e. it can reject, or never terminate).

A NDTM M *operates within time* $f(n)$ if $M(w)$ has depth $\leq f(|w|)$.

Then we can say NDTM M *decides* a language L within time $f(n) if

- M *operates within time* $f(n)
- M accepts L

$L \in NTIME(f(n))$ iff L is decided by some MDTM operating within time $f(n)$.

$$L \in NTIME(f(n))$$

Non-deterministic Polynomial Time (NP)

$$NP = \bigcup_k NTIME(n^k)$$

That is NP is the set of decision problems that can be decided by a NDTM in polynomial time for all inputs.

More usefully: you can "guess" a solution to a NP problem in polynomial time and *verify* that it is a solution in polynomial time.

Simulating a NDTM

Suppose L is decided by some NDTM N in time $f(n)$.
Then it can be simulated by a DTM M in time $O(c^{f(n)})$ for some constant $C > 1$.

We don't know if the simulation can be improved. (i.e. We don't know if $P = NP$.)

NP Problems

- All problems in P
- Integer factorization (most likely in NP)
- Graph Isomorphism
- `SAT`: Given a set of logic clauses with variables, is there an assignment of boolean values to the variables that will satisfy the clauses?
- Hamiltonian Path (`HP`): Given an undirected graph, is there a path visiting each node exactly once?
- Decision version of Travelling Salesman (`TSP(D)`): Is there a route visitng all cities with total distance less than some k?

Reduction

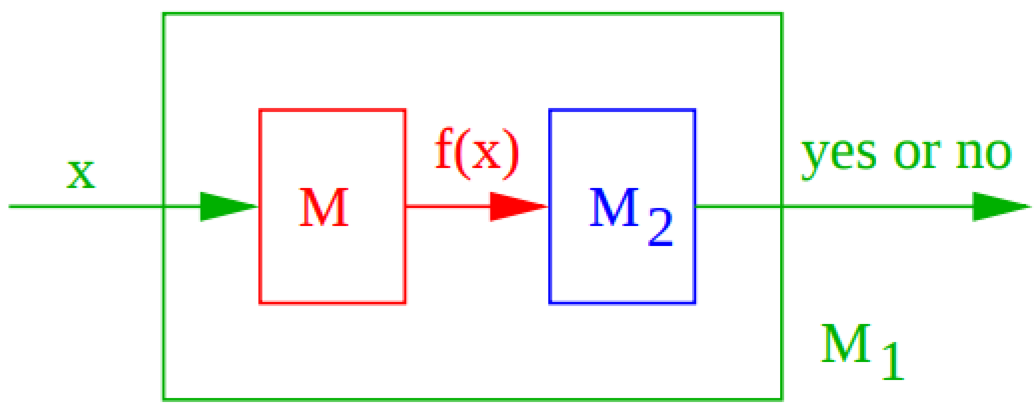
- Let some problem L_1 be *less hard* than some problem L_2.
- We can say $L_1 \leq L_2$
- Two ways: Karp (or many-one) reduction or Cook (or Turing) reduction

(Formal) Karp Reduction

L_1 is Karp reducible to L_2 ($L_1 \leq L_2$) if there is a map f such that

- $x \in L_1$ iff $f(x) \in L_2$ and
- f is in P

Let M_1 decide L_1, M_2 decide L_2 and M computer $f(x)$. Then



Properties of Reduction

If $L_1 \leq L_2$

- if L_2 is in P, then L_1 in P
- if L_2 is in NP, then L_1 in NP
- $\leq$ is transitive

NP-complete (NPC)

L is NP-hard if for some L' in NP, $L' \leq L$.

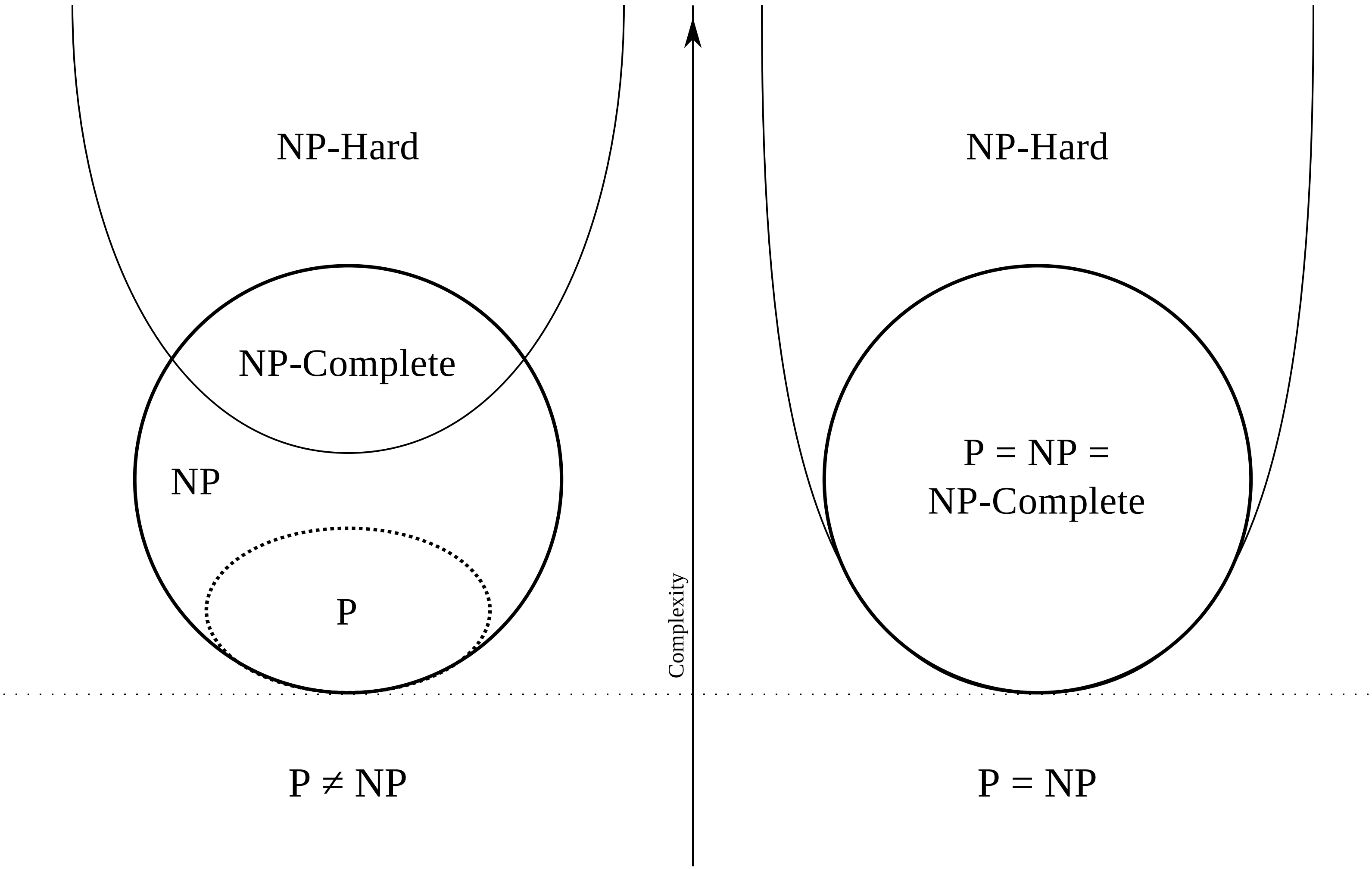
Then

L is NP-complete if

- L is in NP
- L is NP-hard

That is NPC problems are the "hardest" in NP

NPC Relation

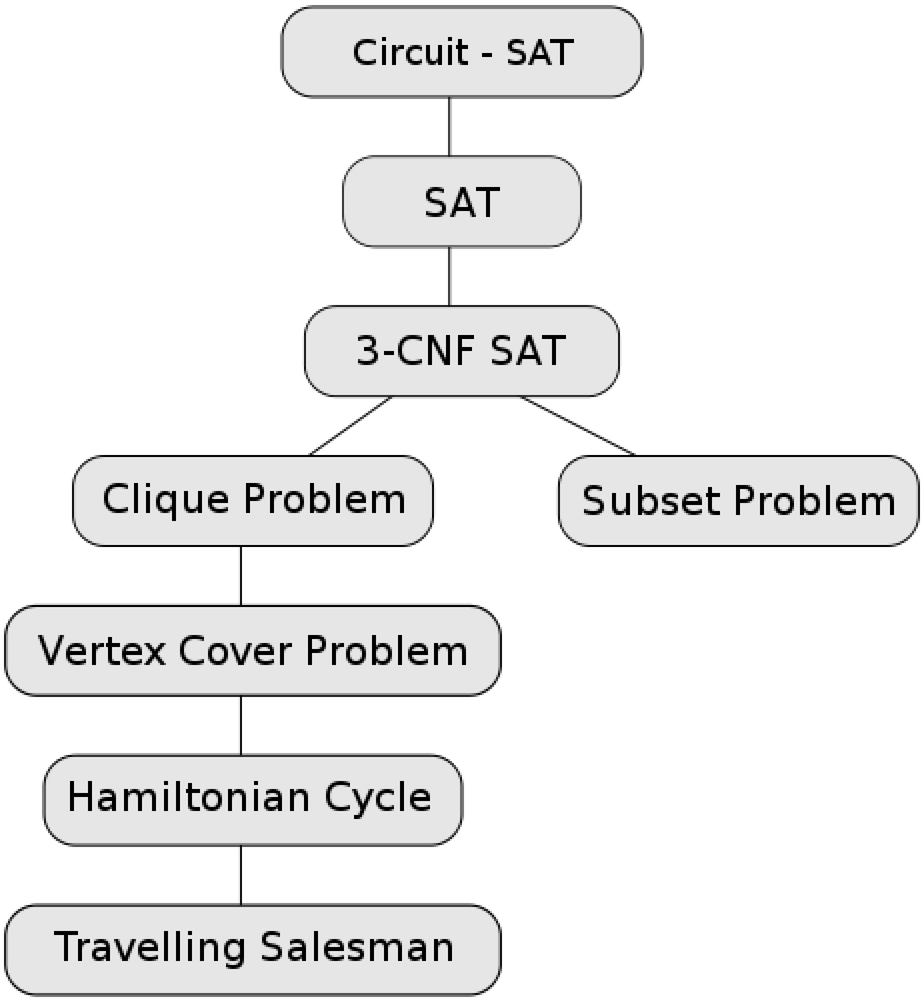


NPC Problems

- `SAT`: Proved by Cook-Levin Theorem
- `TSP(D)`
- Knapsack problem: Can a value of at least V be achieved without exceeding the weight W?
- `HP`
- K-Graph colouring

Huge list of NPC problems

NPC Reduction



P = NP?

- P problems can be solved efficiently.
- NP problems have no known efficient algorithms to solve.

Consequences in Public Key Cryptrography

- Depend on "difficulty" of problems like Discrete logarithm and integer factorization that are known to be neither in P nor NP-complete to create "one-way" functions.
- If $P = NP$, then we can find an effective solution to some NP-complete problem and reduce the rest to that problem to solve them.
- If $P \neq NP$ then we can show that one-way functions exist.

Consequences in Operation

- If $P = NP$ Efficient solution to Integer Linear Programming (for optimization under constraints) and Travelling Salesman