Indian Association for the Cultivation of Science
(Deemed to be University under *de novo* Category)
Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Theory of Computation II: COM 5108

Lecture II

Instructor: Goutam Biswas

Autumn Semester 2023

1 P, NP, coNP and Complete Problems

We have already defined the basic complexity classes **P** and **NP** using polynomial time bounded deterministic and nondeterministic Turing machines. In this section we start with a different definition of the class **NP**. We show that these two definitions coincide. Then following Cook and Levin we establish the existence of an important **NP**-*complete* problem, the *satisfiability* problem of propositional logic.

Many of us have experience that solving a mathematical problem is more difficult than verifying the correctness of a given solution. It seems, it is also manifested in computing (no one knows for sure). There are decision problems for which there is no known *polynomial time* algorithm (Turing machine solving the problem within the number of steps bounded by some polynomial over the length of an input), but if an appropriate "*witness*" or "*proof*" of polynomial size is provided for a *positive answer* ($x \in L$), then that can be verified in polynomial time.

We have already defined $\mathbf{NP} = \bigcup_{k\geq 1} NTIME(n^k)$. Following is an alternate and interesting definition using polynomial time 'proof verifier'.

Definition 1. A language $L \subseteq \{0,1\}^*$ is in **NP**, if there is a polynomial time bounded Turing Machine V called a *verifier* and a polynomial $p(n)$ over $\mathbb{N}_0$, such that

$$x \in L \text{ if and only if } \exists w \in \{0,1\}^{p(|x|)} \text{ such that } V \text{ accepts } < x, w >,$$

where w is called an *witness*, *certificate*, or *proof* of $x \in L$. It is natural that the witness cannot be *too long*. Its length must be polynomial bounded. Otherwise reading the proof will take 'long' time.

If $L \in \mathbf{P}$, then it must be in **NP**, and the verifier is the decider of L with *null string* as the witness. So $\mathbf{P} \subseteq \mathbf{NP}$. The class **coNP** is defined as follows:

$$\mathbf{coNP} = \{L \subseteq \{0,1\}^* : \ \Sigma^* \setminus L \in \mathbf{P}\}^1.$$

[1] The notion of $\Sigma^* \setminus L$ is a bit relaxed. We treate PRIME as a complement of COMPOSITE, and do not bother about 1.

The following proposition shows the equivalence of two definitions of **NP**.

Proposition 1. **NP** as defined above is same as $\bigcup_{k\geq 1}$ **NTIME**(n^k).

Proof: Let L is decided by an NDTM N within $p(n)$ number of steps, where $p(n)$ is a polynomial. So for each $x \in L$, there is a sequence of choices of transitions[2] that leads to an *accept halt*. The length of this sequence cannot be longer than $p(n)$.

Given the input x, the description of N and the choice sequence, a deterministic Turing machine M can verify in polynomial time that x drives N to an *accept halt*. The length of witness, the nondeterministic choice sequence, is bounded by a polynomial, the description of N is fixed (it does not depend on x). So the language L is in **NP**.

Let $L \in$ **NP**, so there is a polynomial time verifier V. Following is the polynomial time NTM.

N: input x

1. Nondeterministically creates a witness string w of the choices of transitions[3]. The length of w is bounded by $p(n)$, where $n = |x|$.

2. Use the verifier V on $< x, w >$.

3. *Accept* if $V(x, w) = 1$, else *reject*.

The running time of N is bounded by polynomial. QED.

A few examples of problems in **NP**.

Example 1.

1. COMPOSITE $= \{n \in \mathbb{N} : \exists p, q \in \mathbb{N} \ (p, q > 1 \text{ and } n = p \cdot q)\}$. The certificate is a pair of factors which cannot be longer than $c\lfloor \cdot \log_2 n\rfloor$, where the input is of length $\lceil \log_2 n\rceil$.
   As a consequence PRIME, the collection of *prime numbers*, is in **coNP**. But now it is known that PRIME is in **P**. So COMPOSITE is also in **P**.

2. Whether a graph G has an *independent set* of certain size.
   INDSET $= \{< G, k >: \ \exists S \subseteq V(G), \ |S| \geq k, \text{ and } \forall u, v \in S, \ \{u, v\} \notin E(G)\}$, $k \in \mathbb{N}$ specifies the size of the *independent set*. The certificate is a set of vertices.

3. TSP $= \{< G = (V, E), d : E \to \mathbb{N}, k >$: there is a travelling salesperson's tour of distance $\leq k\}$. $V = \{1, 2, \cdots, n\}$ is a set of nodes, $\binom{n}{2}$ distances d_{ij} between nodes i and j, and k is a number. Decide whether there is a tour that visits every node exactly once and the total length traversed is at most k. The certificate is a sequence of such nodes.

4. GRAPH-ISO $= \{< G_1, G_2 >$: graph G_1 and graph G_2 are isomorphic $\}$. Given two adjacency matrices E_1 and E_2 corresponding to G_1 and G_2, decide whether there is a permutation $\pi : V_1 \to V_2$ so that E_1 after reordaring is same as E_2.

5. FACTORING $= \{< n, l, u >: \ n, l, u \in \mathbb{N}$, n has a factor p, $l \leq p \leq u\}$. The certificate is p.

[2]In an alternative way we may assume that the degree of nondeterminism is 2, and there two transition functions δ_0 and δ_1.

[3]If $x \in L$, there is a sequence that leads to *accept halt*.

6. PRIME is also in **NP**.
The certificate is more complicated (*Pratt Certificate*). We know that a natural number $p > 1$ is a prime if and only if $\mathbb{Z}_p^*$ is a cyclic group of order $p-1$ i.e. there is an $a \in \mathbb{Z}_p^*$, $1 < a < p$, so that $a^{p-1} \equiv 1(\bmod\ p)$ (*Fermat test*).
And for all prime factors q of $p-1$, $a^{\frac{p-1}{q}} \not\equiv 1(\bmod\ p)$ (*Lucas test*).

   (a) So generator a is part of the certificate. Computation of a^{p-1} modulo p can be performed in $O(l^3)$, where $l = \lceil \log p \rceil$. But it is not sufficient to have a alone as a certificate as it may fail: $4^{15-1} \equiv 1(\bmod\ 15)$, but 15 is no prime. In this case the second test fails as $4^2 \equiv 1(\bmod\ 15)$.
   (b) So the prime factors of $p-1$ is also part of the certificate. But the list of prime factors of $p-1$ may be *false*. Each factor also needs the certificate of its primality. As an example, a false certificate of 45 is $(8; 2, 22)$, where $8^{44} \bmod\ 45 = 1$. Also $8^2 \bmod\ 45 = 19$ and $8^{22} \bmod\ 45 = 19$. This satisfy the second condition also. But if the certificate was correct i.e. $(8; 2, 11)$, the second test will fail as $8^{44/11} \bmod\ 45 = 8^4 \bmod 45 = 1$.
   (c) The complete certificate for a prime defined *inductively* is as follows:
   Basis: $C(2) = ()$, as $2 - 1 = 1$ has no prime factors.
   Induction: $C(p) = (a; q_1, C(q_1), q_2, C(q_2), \cdots, q_k, C(q_k))$, where $q_1, \cdots, q_k$ are prime factors of $p-1$. The process stops at $p = 2$.
   (d) A certificate for 43 is as follows: 2 generates $\mathbb{Z}_{43}^*$ and the prime factors of $43 - 1 = 42$ are $\{2, 3, 7\}$.

$$\begin{aligned} & (2; (2, C(2)), (3, C(3)), (7, C(7))) \\ = \ & (2; (2, ()), (3, (2; (2, ()))), (7, (3; (2, C(2)), (3, C(3))))) \\ = \ & (2; (2, ()), (3, (2; (2, ()))), (7, (3; (2, ()), (3, (2; (2, ())))))) \end{aligned}$$

   (e) It can be proved that the length of the certificate is bounded by $5(\log p)^2$, a polynomial over the length of input.
      i. The bound is true for $p = 2$ and $p = 3$.
      ii. Let the number of prime factors of $p-1$ be $k < \log_2 p$, and they are $q_1 = 2 (p-1 \text{ is even}), q_2, \cdots, q_k$.
      The certificate $C(p) = (a, 2, C(2), q_2, C(q_2), \cdots, q_k, C(q_k))$.
      The length of $C(p)$ is determined by a ($|a| < \log p$), $2k$ separators ($< 2 \log p$), certificate of 2 (of constant length), length of all q_i's ($2 \log p$) (Note[4]), and the lengths of $C(q_i)$'s.
      iii. By the induction hypothesis, the length of the certificate for each q_i is $5(\log q_i)^2$.
      iv. So the total length of the certificate is bounded by

$$|C(p)| \leq 5 \log p + c_1 + 5 \sum_{i=2}^{k} (\log q_i)^2 < 5(\log p)^2.$$

      $\log(q_2 \cdots q_k) \leq \log \frac{p-1}{2} < \log p - 1 \Rightarrow \log q_2 + \cdots + \log q_k < \log p - 1$. Therefore $\log^2 q_2 + \log^2 q_3 + \cdots + \log^2 q_k < (\log p - 1)^2$

$$|C(p)| \leq 5 \log p + 5 \log^2 p - 10 \log p + c < 5 \log^2 p.$$

[4]The number of bits for 2^n is $\log_2 2^n = n$ and the number of bits for n 2's is $2n$. This is the largest possible.

v. The verification of $C(p)$ can be performed in $O(n^4)$ time, where $n = \log p$. The computation of modular exponentiation takes $O(n^3)$ time.

It is not known whether **NP** and **P** are equal. This is considered to be the '*Holy Grail*' of computer science! Most researchers *believe* that $\mathbf{P} \neq \mathbf{NP}$. There are many similar unanswered questions in this area.

The class **NP** was originally defined in terms of nondeterministic Turing machine. The name **NP** comes from *nondeterministic polynomial time bounded Turing machine*.

1.1 NP-hard, and NP-Complete

It was observed that there is a large collection of decision problems (membership in a languages) such as the *satisfiability* of Boolean formula, *independent set* of a certain size in an undirected graph, *3-colouring* of graph etc. are in the class NP. All these problems are difficult to solve in the sense that there is no known polynomial time algorithm. But they have a connection, one of them can be translated to another in polynomial time. The translation or *reduction* is defined in the following way.

Definition 2. A language $L \subseteq \{0,1\}^*$ is *polynomial time mapping reducible* [5] to $L' \subseteq \{0,1\}^*$, if there is a polynomial-time bounded computable function $f : \{0,1\}^* \to \{0,1\}^*$, such that

$$x \in L \text{ if and only if } f(x) \in L', \text{ for all } x \in \{0,1\}^*.$$

This is denoted as $L \leq_P L'$.

Definition 3. A language L' is **NP**-*hard* if for every $L \in \mathbf{NP}$, $L \leq_P L'$. A language L' is called **NP**-*complete* if it is **NP**-*hard* and also belongs to the class **NP**.

Following are a few properties of '$\leq_P$'.

(a) If $L \leq_P L'$ and $L' \in \mathbf{P}$, then $L \in \mathbf{P}$.

(b) $L \leq_P L$, for all L - the binary relation is reflexive.

(c) If $L \leq_P L'$ and $L' \leq_P L''$, then $L \leq_P L''$ - the binary relation is transitive.

(d) What can you conclude about L', if $L \leq_P L'$ and L is **NP**-*hard*?

Proof: Proof of these properties are simple.

(a) Let the polynomial time bounded (p) Turing computable function f reduces L to L' and let L' be decided by a polynomial time bounded (q) Turing machine M. Following is the decider for L.
N: input x

   (i) Compute $f(x)$.
   (ii) Simulate M on $f(x)$.
   (iii) If M accepts $f(x)$, *accept* x;

[5] It is also called *polynomial time many-one reducible*. It was Richard Karp who demonstrated it for the first time [RMK].

(iv) otherwise *reject* x.

(b) Ex.

(c) Ex.

(d) If $L'' \in \mathbf{NP}$, then $L'' \leq_P L \Rightarrow L'' \leq_P L'$ (transitiviti of '$\leq_P$'). So all problems of of $\mathbf{NP}$ are mapping reducible to L', that makes it $\mathbf{NP}$-*hard*.

QED.

Consider the following synthetic language.

$$L_{\mathbf{NP}} = \{< V, x, 1^n, 1^t >:\ \exists u \in \{0,1\}^n \text{ s.t. } V \text{ accepts } < x, u > \text{ within } t \text{ steps}\},$$

where V is an encoding of a deterministic Turing machine.

Proposition 2. L_{NP} is $\mathbf{NP}$-*complete*.

Proof:

$L_{\mathbf{NP}}$ is in $\mathbf{NP}$:
We design a verifier V' for $L_{\mathbf{NP}}$. Consider $< V, x, 1^n, 1^t >$, if there is an $u \in \{0,1\}^n$ such that V accepts $< x, u >$ in time t, then $< V, x, 1^n, 1^t > \in L_{\mathbf{NP}}$. This u can be used as a certificate of $< V, x, 1^n, 1^t >$. Its length is linear with respect to the length of input, due to 1^n. V' simulates V on $< x, u >$ for at most t steps. This can be done in polynomial time. If V *accepts*, then V' returns 'Y' else it returns 'N'.

Any language $L \in \mathbf{NP}$ is polynomial time reducible to $L_{\mathbf{NP}}$:
If L is in $\mathbf{NP}$, then by definition there is a polynomial $p : \mathbb{N}_0 \to \mathbb{N}_0$ and a polynomial time bounded Turing machine V' so that for all $x \in \{0,1\}^*$, $x \in L$ if and only if there is a $u \in \{0,1\}^{p(|x|)}$ such that V' accepts $< x, u >$ in polynomial time. Let the running time of V' be bounded by the polynomial $q : \mathbb{N}_0 \to \mathbb{N}_0$. The reduction is

$$x \mapsto < V', x, 1^{p(|x|)}, 1^{q(|x|+p(|x|))} > .$$

This mapping can be done in polynomial time as $< V' >$ is of fixed length, and lengths of both $1^{p(|x|)}$ and $1^{q(|x|+p(|x|))}$ are polynomial bounded. QED.

But this language is not very interesting or useful for reducing problems from different practical domains. S A Cook in 1971 [SAC] and L A Levin in 1973 (independently from USSR) [LAL] presented the notion of $\mathbf{NP}$-*completeness* and gave examples of $\mathbf{NP}$-complete problems from domains like logic etc.. Subsequently R M Karp in 1972 [RMK] showed a large collection of practical problems to be $\mathbf{NP}$-*complete*.

1.2 Boolean Formula

Definition 4. A *boolean formula* is defined as follows.

1. Boolean constants *true* and *false* are boolean formulas.

2. Boolean variables $x_1, x_2, \cdots$ (that takes values *true* or *false*) are boolean formulas.

3. If f_1 and f_2 are boolean formula, then so are $(f_1 \vee f_2)$, $(f_1 \wedge f_2)$ and $\neg f_1$.

4. Nothing else is a boolean formula.

A variable or a negation of a variable is called a *literal*. We shall use $\overline{f}$ for $\neg f$ for negation of a formula.

We encode *true* and *false* as 1 and 0 respectively. If ϕ is a boolean formula of n variables, $x_1, \cdots, x_n$, we can assign truth values to the variables (an element $v \in \{0,1\}^n$) and get a truth value $\phi(v)$ for the formula. All possible assignments to the variables form the truth table. A boolean formula ϕ is *satisfiable* if there is a truth assignment that make ϕ *true*. It is *unsatisfiable* if for no truth assignment the formula is *true*.

Example 2. The formula

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3})$$

is satisfiable with assignment $x_1 = 1, x_2 = 0, x_3 = 1$. But following formulas are unsatisfiable

1. $x \wedge \overline{x}$.
2. $(x_1 \vee x_2) \wedge \overline{x_1} \wedge \overline{x_2}$.
3. $(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_1})$.

A *boolean formula* is in *conjunctive normal form (CNF)* or *clausal normal form* if it is *conjunction (and)* of *clauses*. A *clause* is a *disjunction (or)* of *literals*. It is in *disjunctive normal form (DNF)* or *sum of product* forn if it is *disjunction* of the *conjunctions* of *literals*. It is a *k-CNF* if it is in CNF and every *clause* has at most k literals.

We define the following languages:

$$SAT^6 = \left\{ \phi = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n_i} u_{ij} \right) : \ m, n_i > 0 \ \text{ and } \phi \text{ is satisfiable} \right\},$$

where u_{ij} is a *literal* and

$$3SAT = \left\{ \phi = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{3} u_{ij} \right) : \ m > 0 \text{and } \phi \text{ is satisfiable} \right\},$$

$$2SAT = \left\{ \phi = \bigwedge_{i=1}^{m} (l_{i1} \vee l_{i2}) : \ \phi \text{ is satisfiable} \right\},$$

Proposition 3. Any Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be expressed as a *disjunctive normal form (DNF)* or a *conjunctive normal form (CNF)* (functional completeness of $\vee, \wedge, \neg$).

Proof: It is known that for a n variable formula there are 2^n rows in the truth table. We take the standard convention that the truth assignment corresponding of the variables in the j^{th} row is the n-bit binary representation of j, $0 \leq j \leq 2^n - 1$.

Consider the truth-table corresponding to an n-variable Boolean function $f(x_1, \cdots, x_n)$. For the equivalent DNF formula ψ, we only consider those rows of the table where the truth values of the function is 1. Each row corresponds to

[6] One may define $SAT = \{\phi : \ \phi \text{ is satisfiable.}\}$.

a *conjunction* ($\wedge$) of literals, and all of them are connected by disjunction ($\vee$) to form the final formula ψ. Let j be one such row and the values of the variables be $v_1, \cdots, v_n$ (an an *n-tuple* of 1's and 0's). Corresponding to this row, the conjunct of literals is $D_j = l_1 \wedge \cdots \wedge l_n$, where $l_i = x_i$ if $v_i = 1$, otherwise it is $\overline{x_i}$. It is clear that no other assignment of variables can make D_j *true* as at least one of the literals will be *false*. Finally the equivalent DNF formula is

$$\psi(x_1, \cdots, x_n) = D_{i_1} \vee \cdots \vee D_{i_k},$$

where there are k rows with truth values 1.
We observe that $\psi(v_1, \cdots, v_n) = 1$, if one of disjuncts is 1 i.e. one of the rows of the truth table of ψ has a 1.
On the other hand if $\psi(v_1, \cdots, v_n) = 0$, then all D_{i_j}s are *false* or 0. So $f(v_1, \cdots, v_n) = 1$ if and only if $\psi(x_1, \cdots, x_n) = 1$.

As an example consider a 5-variable formulas and the j^{th} row of the truth table where the truth value 1. Let the values of the Boolean variables in the j^{th} row be (01101), then the corresponding conjunctive formula $D_j = \overline{x_1} \wedge x_2 \wedge x_3 \wedge \overline{x_4} \wedge x_5$. It is clear that $D_j(01101) = 1$, but $D_j(k) = 0$ for any other truth assignment.

Similarly to get the equivalent CNF formula of f, we consider only those rows where the truth values are 0. If the values of the variables $(x_1, \cdots, x_n)$ are $(v_1, \cdots, v_n)$ in one such row j, we take $C_j = l_1 \vee \cdots \vee l_n$, where $l_i = x_i$ if $v_i = 0$, otherwise it is $\overline{x_i}$. The truth value of C_j is 1 or *true* if the value of one variable say, x_i, is changed to $1 - v_i$.

The equivalent CNF formula of f is

$$\psi(x_1, \cdots, x_n) = C_{i_1} \wedge \cdots \wedge C_{i_k},$$

where there are k rows of the truth table with the truth values 0.

As an example we consider a 5-variable Boolean formula. Let the variables in the j^{th} row of the truth table, where $\phi(j) = 0$, takes the values (10011), then $C_j = \overline{x_1} \vee x_2 \vee x_3 \vee \overline{x_4} \vee \overline{x_5}$. It is clear that $C_j(10011) = 0$, but $C_j(k) = 1$, if $k \neq 10011$. QED.

The length of such a formula may be $O(n2^n)$, where the length of a formula is the count of the number of $\vee$ and $\wedge$. The size of a truth table is exponential in the number of variables.

Before we prove that 3SAT is **NP**-complete, we shall prove an interesting result that is 2SAT $\in$ **P**.

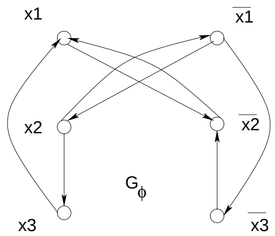
Let ϕ be a 2SAT formula. We construct a graph $G_\phi = (V_\phi, E_\phi)$, where $V_\phi = \{x, \overline{x} :\ x \text{ is a variable in } \phi\}$, and $E_\phi = \{(l_1, l_2) : \text{if } (l_2 \vee \overline{l_1}) \text{ (or } (\overline{l_1} \vee l_2)) \text{ is a clause in } \phi\}$.
Each edge in G_ϕ captures a clause in ϕ as a logical implication. Note that $(v \vee \overline{u})$, $(\overline{u} \vee v)$ and $(u \Rightarrow v)$ are logically equivalent.
There is a symmetry in the graph. A clause $(l_1 \vee l_2) = (\overline{\overline{l_1}}, \overline{\overline{l_2}})$ gives rise to two edges: $(\overline{l_1} \vee l_2)$ and $(\overline{l_2}, l_1)$. If there is a path from some literal $l_1 \rightarrow \cdots \rightarrow l_k$, $k \geq 1$ in the graph, then by the transitivity of implication we have $(l_1 \Rightarrow l_k)$. If there is a path from l_1 to l_k, then there is a path from $\overline{l_k}$ to $\overline{l_1}$.

Following the semantics of implication, if l_1 is assigned the value *true*, then every literal reachable from l_1 in G_ϕ should also be *true*. Symmetrically, if l_1 is assigned *false*, then all its predecessor literals will also be *false*.

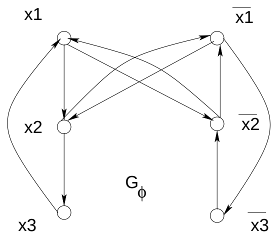
A variable x cannot be assigned any truth value in a formula ϕ, if there is a path from x to $\overline{x}$ (equivalently a path from $\overline{x}$ to x) in G_ϕ, as it is same as $x \Leftrightarrow \overline{x}$ - a contradiction.

Example 3. Consider the following example,

$$\phi = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_2} \vee x_3).$$



There is an satisfying assignment, $x_1 = 1, x_2 = 0, x_3 = 1$. But if we include another clause, $(\overline{x_1} \vee x_2)$, then there is no satisfying assignment any more, as there will be a path from x_1 to $\overline{x_1}$ and also a path from $\overline{x_1}$ to x_1.



Lemma 1. A 2SAT formula ϕ is *unsatisfiable* if and only if there is a variable x such that there is a path from x to $\overline{x}$ (also a path from $\overline{x}$ to x).

Proof: Let for some variable x there are two such paths and at the same time ϕ is satisfiable. So there is a truth value $v(x)$ for x. Let $v(x)$ is *true* and $v(\overline{x})$ is *false*. As there is a path from x to $\overline{x}$, there must be an edge (l_1, l_2) such that $v(l_1) =$ *true* but $v(l_2) =$ *false*. The corresponding clause is $(\overline{l_1} \vee l_2)$ and is not satisfiable - a contradiction. Similar argument works for $v(\overline{x}) =$ *false*.

In the other direction, we assume that there is no variable x with such pair of paths. The satisfying truth assignment of ϕ is as follows:

The following procedure will be repeated until all nodes are assigned truth values.

Take a literal l, a node in G_ϕ, that has not been assigned any truth value and there is no path from l to $\overline{l}$. Assign *true* to l and every literal reachable from the node of l. Assign *false* to the negation of these literals. In other words, if a node is assigned *false* then its predecessor is also assigned *false*. If l' is reachable from l, then $v(l') =$ *true*. If the node $\overline{l}$ is reachable from $\overline{l'}$, then both have value *false*.

We claim that the process cannot assign same truth value to l' and $\overline{l'}$ i.e. nodes of both l' and $\overline{l'}$ cannot be reachable from l. If that was possible then $\overline{l}$ would have been reachable from both of them resulting a path from l to $\overline{l}$. QED.

Proposition 4. 2SAT $\in$ **P**

Proof: The steps of the algorithm are as follows:

M: input ϕ

1. Build the graph G_ϕ.
2. For each variable x, test whether $\overline{x}$ is reachable and vice versa.
3. *Accept* if no such path exist; otherwise *reject*.

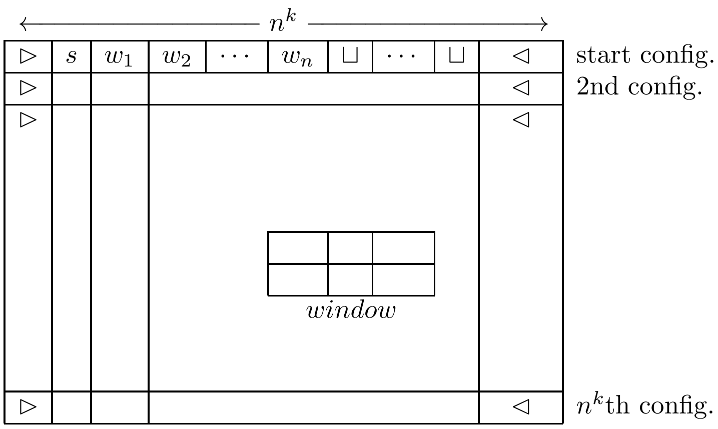
It is an $O(n^2)$ algorithm. QED.

It is interesting that 2SAT is of so low complexity, but there is no known polynomial time algorithm for 3SAT. It fact there is a very strong belief that it is impossible to have one.

1.3 Cook-Levin Theorem

Cook [SAC] and Levin [LAL] demonstrated the first **NP**-complete problem.
Theorem 2. Both SAT and 3SAT are **NP**-*complete* (Cook and Levin).
Lemma 3. Both SAT and 3SAT are in NP.
Proof: The certificate is the truth value assignment of the variables in the SAT (3SAT) formula ϕ. Given an assignment it is possible to evaluate the truth value of ϕ in polynomial time. QED.
Theorem 4. SAT is **NP**-complete.

We need to reduce any language $L \in \mathbf{NP}$ to SAT in polynomial time. Let L is decided by an NTM N in polynomial time. The reduction of L to SAT takes a $x \in \Sigma^*$ as input and produces a boolean formula ϕ that in a sense simulates the computation of N on the input x. If N *accepts* x i.e. $x \in L$, then there is a *satisfying* truth assignment for ϕ. Otherwise ϕ is *unsatisfiable*.
Proof: Let N decides L in n^k time. The total computation of N on the input $x = w_1 w_2 \cdots w_n$ can be captured by a table of size $n^k \times n^k$.



QED.

We have used two end markers $\{\triangleright, \triangleleft\}$ for every configuration. The first row is the *start configuration* of the computation on input $x = w_1 w_2 \cdots w_n$ at the start state s. The table corresponding to an input $x \in L$ must have a row of *accepting* configuration. The problem is to determine whether there is a table with an *accepting* configuration corresponding to the nondeterministic computation of N on x.

The reduction maps $x \mapsto \phi$. The variables of the boolean formula ϕ are defined as follows:

Let $N = (Q, \Sigma, \delta, s)$ and $\Sigma = \{\rhd, \sqcup, s, \cdots\}$. For each $p \in C = Q \cup \Sigma \cup \{\lhd\}$ and $1 \leq i, j \leq n^k$ (row and column), we have a variable $v_{i,j,p}$.

A cell at the i^{th} row and j^{th} column is $cell[i, j]$. A variable $v_{i,j,p}$ for the $cell[i, j]$ is 1 (*true*) if its content is p, otherwise it is 0 (*false*).

The formula ϕ is a conjunction of four formulas:

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Each $cell[i, j]$ contains exactly one $p \in C$.

$$\phi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} \left(\left(\bigvee_{p \in C} v_{i,j,p} \right) \wedge \left(\bigwedge_{\substack{p,q \in C, \\ p \neq q}} (\overline{v_{i,j,p}} \vee \overline{v_{i,j,q}}) \right) \right).$$

Example 4. Let v_1, v_2, v_3 be boolean variables. The following formula is true if and only if exactly one variable is true.

$$f(v_1, v_2, v_3) = (v_1 \vee v_2 \vee v_3) \wedge (\overline{v_1} \vee \overline{v_2}) \wedge (\overline{v_2} \vee \overline{v_3}) \wedge (\overline{v_3} \vee \overline{v_1}).$$

The *start configuration* of N on input $x = w_1 w_2 \cdots w_n$ is proper if the following formula is *satisfiable*.

$$\phi_{start} = v_{1,1,\rhd} \wedge v_{1,2,s} \wedge v_{1,3,w_1} \wedge \cdots v_{1,n+2,w_n} \wedge v_{1,n+3,\sqcup} \wedge \cdots \wedge v_{1,n^k-1,\sqcup} \wedge v_{1,n^k,\lhd}.$$

There is an *accepting configuration* in the table of computation of N if the following formula is satisfied.

$$\phi_{accept} = \bigvee_{1 \leq i,j \leq n^k} v_{i,j,Y}.$$

The transition from configuration C_i to C_{i+1} must be compatible with the state transition relation of N, for all i, $1 \leq i < n^k$. This is ensured by the *satisfiability* of ϕ_{move}.

At every point in time the computation of a TM is *local*. The head can move one place to left or to right, or it may remain stationary after changing the content of the current cell. The validity of $C_i \rightarrow_N C_{i+1}$ is checked by looking at every window of size 2×3 on these pair of configurations.

Given an NTM N, there is a finite set of *valid windows* that are *compatible* to Q, Σ and Δ.

Example 5. Let $((p, a), \{(p, b, \rightarrow)\}), ((p, b), \{(q, c, \leftarrow), (q, a, -)\}) \in \Delta$.
The state in the following positions can affect the *window*.

		Window			
1	p				
2		p			
3			p		
4				p	
5					p

Following are a few possible valid windows. α, β are any tape symbols.

1(a)

a	α	β
p	α	β

, 1(b)

b	α	β
c	α	β

, 1(c)

b	α	β
a	α	β

, central cells are unchanged.

$$2(a)\begin{array}{|c|c|c|}\hline p & a & \alpha \\ \hline b & p & \alpha \\ \hline\end{array},\quad 2(b)\begin{array}{|c|c|c|}\hline p & b & \alpha \\ \hline \beta & c & \alpha \\ \hline\end{array},\quad 2(c)\begin{array}{|c|c|c|}\hline p & b & * \\ \hline q & a & * \\ \hline\end{array},$$

And there are many more but finite and depends on N but not on input x.

Following are a few invalid windows.

$$3(a)\begin{array}{|c|c|c|}\hline \alpha & a & \beta \\ \hline \alpha & b & \beta \\ \hline\end{array},\quad 3(b)\begin{array}{|c|c|c|}\hline p & a & \alpha \\ \hline p & b & \alpha \\ \hline\end{array},\quad 3(c)\begin{array}{|c|c|c|}\hline \alpha & p & b \\ \hline q & \alpha & a \\ \hline\end{array}.$$

Basis: The formula ϕ_{start} is satisfiable if and only if the first row of the table is a start configuration.

Hypothesis: C_i is a reachable configuration.

Induction: If all windows of (C_i, C_{i+1}) are valid, then C_{i+1} is also a reachable configuration i.e. $C_i \to_N C_{i+1}$.

We call an window W_{ij} if the *cell*$[i, j]$ is in its upper central position. In W_{ij}, if upper three symbols are tape symbols, then the content of *cell*$[i, j]$ (upper-central) is same as the content of *cell*$[i + 1, j]$ (central-lower). The central cell does not change if there is no adjacent state symbol.

If a W_{ij} contains a state symbol in *cell*$[i, j]$ (top-center), it is guaranteed that the lower three cells are updated properly following the transition relation of N.

$$\phi_{move} = \bigwedge_{\substack{1 \le i < n^k \\ 1 < j \le n^k}} \text{valid } W_{ij}.$$

Each valid W_{ij} can be replaced by the content of its cells. Let the possible contents of 6-cells be $a_1, \cdots, a_6$. The "valid W_{ij}" can be replaced by

$$\bigvee_{\text{valid } a_1,\cdots,a_6} (v_{i,j-1,a_1} \wedge v_{i,j,a_2} \wedge v_{i,j+1,a_3} \wedge v_{i+1,j-1,a_4} \wedge v_{i+1,j-,a_5} \wedge v_{i+1,j+1,a_6})$$

The time complexity of the reduction is as follows:

- The variables are of the form $v_{i,j,p}$, where $1 \le i, j \le n^k$, $p \in C = Q \cup \Sigma \cup \{\lhd\}$. The number of variables are $|C| \times n^k \times n^k = O(n^{2k})$ as $|C|$ does not depend on the length of the input. Lengths of i, j and p takes $2k \log n + \log p = O(\log n)$ bits. There is a length of $O(\log n)$ bits for each variable.
- The formula for the validity of cells, ϕ_{cell} is a conjunction over $n^k \times n^k$ cells. The length of each conjuncts is independent of the length of input x. So the the length of ϕ_{cell} is $O(n^{2k})$.
- The formula ϕ_{start} encodes the first row with n^k variables and $n^k - 1$ '$\wedge$' operators. Its length is $O(n^k \log n)$. The contribution
- The formula ϕ_{accept} is a disjunction over all cells. Its length is $O(n^{2k} \log n)$.
- Similarly the formula for moves, ϕ_{move} is over all windows, over all (almost) cells is also $O(n^{2k} \log n)$. The number of valid windows is independent of the length input x.

The total length of the formula is $O(n^{2k} \log n)$. The claim that it can be generated in polynomial time due to its *repetitive nature*!

1.4 Reduction

We have already proved that 3SAT $\in$ **NP**. Now we reduce **SAT** to **3SAT** in polynomial time to show the following.

Proposition 5. 3SAT is **NP**-hard

Proof: Following is a reduction of SAT to 3SAT. Consider a CNF formula $\phi = C_1 \wedge \cdots \wedge C_k$. We wish to transform it in equivalent 3CNF formula ψ. Let the clause C_i has $m > 3$ literals i.e. $C_i = l_{i1} \vee \cdots \vee l_{im}$. We introduce a new variable z_{i1} and write $f_1 = (l_{i1} \vee \cdots l_{i(m-2)} \vee z_{i1}) \wedge (\overline{z_{i1}} \vee l_{i(m-1)} \vee l_{im})$. If there is an assignment that makes C_i *false* (all its literals are *false*), then no assignment of z_{i1} can make f_1 *true*. On the other hand, if there is an assignment that makes C_i *true*, then there is an assignment of z_{i1} that makes f_1 *true*. If in the given satisfying assignment both $l_{i(m-1)}$ and l_{im} are false then $z_{i1} \leftarrow 0$, else $z_{i1} \leftarrow 1$.

This process increases the length of the formula by 4 (increase in the number of $\vee$ and $\wedge$) and reduce the clause size to $m-1$. If the transformation is repeated for $m-3$ times, the increase in length is by $4(m-3)$. QED.

We reduce 3SAT to the following set to prove that it is **NP**-hard.

Proposition 6.

$$INDSET = \{< G, k >: \exists S \subseteq V(G) \text{ s.t. } |S| \geq k \text{ and } \forall u, v \in S, \{u, v\} \notin E(G)\}$$

is **NP**-*complete*.

Proof: We show two different reductions.

First reduction: Let there be m number of 3-literal clauses in the 3CNF Boolean formula. Each clause C gives a *triangle* T with the vertices labelled by the literals. If two clauses C_i and C_j has a variable x_k and its negation $\overline{x_k}$, we join the corresponding vertices by an edge (edge for inconsistency).

If there is a satisfying assignment $v : Var \rightarrow \{0, 1\}$, then each clause is also satisfied, so there is a vertex in each triangle whose literal value is 1. These m vertices will form an independent set. There cannot be any edge between a pair of such vertices. An edge between two triangles is between a variable and its negation.

We cannot form an independent set by taking two vertices from a triangle. Also we cannot take two vertices of two triangles that are connected by an age (*inconsistent*). So, if there is an independent set of size m, assigning 1 to corresponding literals gives a *satisfying* assignment. There may be some extra variables, that can be assigned any value.

Second Reduction: Associate a complete graph of 7 vertices to every clause. So there are $7m$ vertices. Among the eight possible assignments, $\{000, \cdots, 111\}$, one will make a clause false e.g. if the clause is $x_1 \vee \overline{x_4} \vee \overline{x_{11}}$, then the assignment $x_1 \leftarrow 0,\ x_4 \leftarrow 1,\ x_{11} \leftarrow 1$ will make it *false*. Associate remaining seven satisfying assignments to seven nodes of the clause. If two nodes in two different clauses have a common variable assigned to different values, 0 and 1, join them by an edge (inconsistency).

If there is a satisfying assignment $v : Var \rightarrow \{0, 1\}$ of ϕ, then pick-up a vertex from the seven nodes of a clause C which has the restriction of v to the variables of the clause. This selected vertex cannot have any edge going out of the clause (7-node complete subgraph) to another selected vertex of a different clause, as they are selected using a satisfying assignment. So there is an independent set of size m.

An independent set cannot take more than one vertex from the 7-vertices of any clause. If there is an independent set of size m, their vertices are coming from m different clauses. There cannot be any edge between these vertices as they form an independent set. The assignment given to the variables *locally* to every clause gives a consistent *global* assignment. As an example corresponding to the clause $x_1 \vee \overline{x_4} \vee \overline{x_{11}}$, if the vertex with the assignment $x_1 \leftarrow 1,\ x_4 \leftarrow 0,\ x_{11} \leftarrow 1$, is an element of the independent set, then there is a satisfying assignment that is an extension of this local assignment. QED.

Example 6. Consider the following 3SAT formula and show both the reductions.

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

Proposition 7. SUBSET-SUM $= \{< S, t >:\ S = \{x_1, \cdots, x_k : x_i \in \mathbb{N}\}$ is a multiset and for some $\{y_1, \cdots, y_l\} \subseteq S,\ \sum_{i=1}^{l} = t\}$. is **NP**-complete.

Proof: The certificate for SUBSET-SUM is C, a collection of elements of S. Following is a verifier.

$V =$ "On input $<< S, t >, C >$

1. Test whether $C \subseteq S$.
2. Test whether $\sum C = t$.
3. *Accept* if both are *true*, else *reject*."

So SUBSET-SUM $\in$ **NP**.

We reduce a 3SAT formula ϕ to an instance of a SUBSET-SUM problem in polynomial time to show that it is **NP**-complete.

Let the variables of ϕ be $x_1, \cdots, x_l$ and the clauses be $C_1, \cdots, C_k$. Following table shows the elements of S and the value t constructed from the formula ϕ such that $< S, t > \in$ SUBSET-SUM if and only if ϕ is satisfiable.

Table (T)

	Variables						*Clauses*				
	x_1	x_2	x_3	x_4	$\cdots$	x_l	c_1	c_2	c_3	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	1	1	$\cdots$	0
y_2		1	0	0	$\cdots$	0	0	1	1	$\cdots$	0
z_2		1	0	0	$\cdots$	0	1	0	0	$\cdots$	0
y_3			1	0	$\cdots$	0	0	0	0	$\cdots$	0
z_3			1	0	$\cdots$	0	1	1	1	$\cdots$	0
$\vdots$	$\vdots$		$\vdots$		$\vdots$			$\vdots$		$\vdots$	
y_l					$\cdots$	1	0	0	0	$\cdots$	0
z_l					$\cdots$	1	0	0	0	$\cdots$	0
g_1					$\cdots$	0	1	0	0	$\cdots$	0
h_1					$\cdots$	0	1	0	0	$\cdots$	0
g_2					$\cdots$	0	0	1	0	$\cdots$	0
h_2					$\cdots$	0	0	1	0	$\cdots$	0
g_3					$\cdots$	0	0	0	1	$\cdots$	0
h_3					$\cdots$	0	0	0	1	$\cdots$	0
$\vdots$	$\vdots$		$\vdots$		$\vdots$			$\vdots$		$\vdots$	
g_k					$\cdots$	0	0	0	0	$\cdots$	0
h_k					$\cdots$	0	0	0	0	$\cdots$	0
t	1	1	1	1	$\cdots$	1	3	3	3	$\cdots$	3

Each row of the table (other than t) corresponds to a decimal number, member of S. These decimal numbers use digits 0 and 1. The decimal number t uses digits 1 and 3. Blanks correspond to zeros.

(a) For each variable x_i there are a pairs of numbers y_i and z_i. The digits of each of them is partitioned in to two parts, the *variable* part (left side) and the *clause* part.

(b) The digit in $T[y_i, x_i] = T[z_i, x_i] = 1$. All other digits in the *variable* part are 0's. We select y_i from S if the truth value of $x_i \leftarrow 1$. Otherwise select z_i.

(c) The digits in $T[y_i, c_j] = 1$ if the clause c_j has the literal x_i. The digit in $T[z_i, c_j] = 1$ if the clause c_j has the literal $\overline{x_i}$. Other digits are 0's.

(d) S also contains a pair of numbers g_j and h_j for each clause c_j. The digit in $T[g_j, c_j] = T[h_j, c_j] = 1$. All other digits of these numbers are 0's.

(e) The digits in the *variable* part of t are all 1's and the digits in the *clause* part of t are all 3's.

(f) The target is to get the value of t after adding the selected numbres y_i or z_i for $i = 1, 2, \cdots, l$ (each variable) and zero, one or both of g_j, h_j for $j = 1, 2, \cdots, k$ (each clause).

Consider $\phi = C_1 \wedge C_2 \wedge C_3$ where $C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$, $C_2 = \overline{x_1} \vee x_2 \vee x_3$ and $C_3 = \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$. A satisfying assignment is $x_1 \leftarrow 1$, $x_2 \leftarrow 1$, and $x_3 \leftarrow 0$. We choose y_1, y_2, z_3 (ignore other rows and columns of the table). So far the sum is $100100 + 010011 + 001111 = 111222$. We also choose g_1, g_2, g_3 to make the final sum equal to 111333 (t).

If ϕ is satisfiable: there is a truth assignment for each variable. If $x_i \leftarrow 1$, we choose the number y_i. If $x_i \leftarrow 0$, we choose thenumber z_i. Whatever be the case, when added we get 1 in first l digits of t.
At least one of the three literals of a clause C_j must be true. It may be due to l_i. If $l_i = x_i$ i.e. $x_i \leftarrow 1$, we have already chosen y_i which has 1 in its c_j column. If $l_i = \overline{x_i}$, we have chosen z_i and it has 1 in its c_j column. The sum of the digits of the column c_j for a satisfying assignment can be 1, 2, or 3. They can all be brought to 3 by adding g_j, h_j. But that is not possible if a clause is unsatisfiable.

If subset of S gives the sum t: for every i either y_i or z_i is chosen, but not both, as first l digits of t are all 1's. In column c_j at most 2 can be supplied from g_j and h_j. So 1 must come from the literal of a clause. So the clause is satisfied. QED.

Proposition 8. 3COL = $\{< G >$: graph G has a vertex colouring with at most three colours$\}$ is **NP**-*complete*.

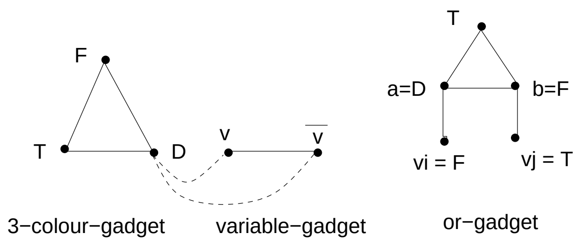
Proof: The certificate of 3COL is colouring of different vertices. A polynomial time verifier can check validity of colouring in polynomial time. So 3COL is in **NP**.

We reduce 3SAT to 3COL. Let ϕ be a 3CNF formula with m clauses $c_1, \cdots, c_m$ and n variables $x_1, \cdots, x_n$. The construction is as follows:

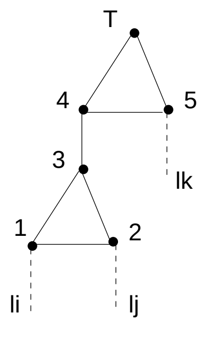
1. There is a pair of vertices $v_i, \overline{v_i}$ for every variable x_i and its negation $\overline{x_i}$.

2. Five vertices $u_{i1}, \cdots, u_{i5}$ for each clause c_i.

3. Three special vertices T, F, D for three colours *true*, *false* and D.



Form a triangle with T, F, D to force three colours to colour them.
Form a triangle with $v_i, \overline{v_i}$ and D so that a variable can take either colour T (*true*) or F (*false*) and not D.
The difficult part is to ensure that at least one literal in every clause is *true* if and only if the graph is 3-colourable.
We start with a graph of 3-vertices, a, b,and T forming a triangle. The vertex a is connected to a literal-vertex v_i and b to a literal-vertex v_j. In the triangle of a, b, T, a and b can be coloured with F and D only. Literal vertices can be coloured only with T and F. So one literal must be coloured with T. This is called an "or gadget".
Now we look into the five vertices $u_{i1}, \cdots, u_{i5}$ corresponding to the clause c_i. The corresponding graph has following edges: $\{\{u_{i1}, u_{i2}\}, \{u_{i1}, u_{i3}\}, \{u_{i2}, u_{i3}\}\}$, $\{\{u_{i3}, u_{i4}\}, \{u_{i4}, u_{i5}\}, \{u_{i5}, T\}, \{u_{i4}, T\}\}$ and $\{\{u_{i1}, l_i\}, \{u_{i2}, l_j\}, \{u_{i5}, l_k\}\}$, where v_i, v_j, v_k are are vertices corresponding to literals l_i, l_j, l_k respectively.

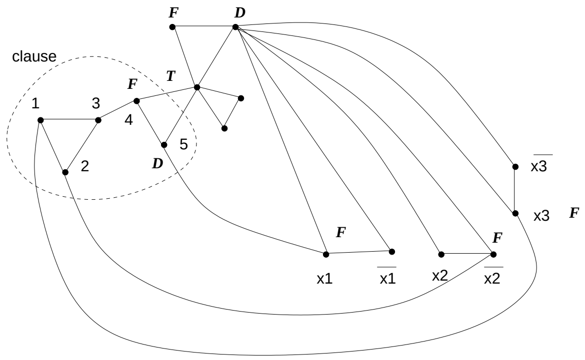


Following are the possible colour assignments:

u_{i5}	u_{i4}	u_{i3}	u_{i1}	u_{i2}	Literal coloured T
F	D				l_k
D	F	T	F	D	l_i
D	F	T	D	F	l_j
D	F	D	T	F	l_j
D	F	D	F	T	l_i

So one literal must be coloured T. The claim is that 3CNF formula ϕ is satisfiable if and only if the graph is 3-colourable.

Following figure shows an example with a clause $C = x_1 \vee \overline{x_2} \vee x_3$.



If all three literals are *false*, then node 1 and 2 are coloured with T and D. But that needs a 4th colour for node 3. But the table shows that if one of the literal is *true* i.e. coloured with T. then the graph is 3-colourable. QED.

Proposition 9. dHAMPATH $= \{< G, s, d >: G$ is a directed graph with a Hamiltonian path from s to $d\}$.

Proof: It is clear that dHAMPATH is in **NP**. A sequence of vertices on the path is a certificate. This can be verified in polynomial time.

We reduce 3SAT[7] to dHAMPATH in polynomial time. Consider a 3CNF formula with m clauses and n variables, $x_1, \cdots, x_n$.

$$\phi = (l_{11} \vee l_{12} \vee l_{13}) \wedge \cdots \wedge (l_{m1} \vee l_{m2} \vee l_{m3}).$$

where l_{ij}, $1 \leq i \leq m$ and $1 \leq j \leq 3$, is x_k or $\bar{x_k}$, for some k, $1 \leq k \leq n$.

There is a starting vertex labelled with s and an end vertex labelled with d.

For every variable x_k there is a *doubly linked chain-graph* of $3m + 1$ vertices.

There is a vertex $s_{i(i+1)}$ between every pair of doubly linked chain-graphs corresponding to variables x_i and x_{i+1}, $1 \leq i < n$. There are directed edges, from s to the two ends of the *doubly linked graph* of x_1, from $s_{i(i+1)}$ to the two ends of the *doubly linked graph* of x_{i+1}, from the two ends of the *doubly linked graph* of x_i to $s_{i(i+1)}$, $1 \leq i < n$, and from two ends of the *doubly linked graph* of x_n to d.

For every clause there is a vertex. Call them $c_1, \cdots, c_m$. Each doubly linked graph corresponding to a variable has a pair of nodes corresponding to a clause. Every such pair is separated by a node, and there are two terminal nodes. This accounts for the number $3m + 1$,

$$\bigcirc \bigcirc_1 \bigcirc_1 \bigcirc \bigcirc_2 \bigcirc_2 \bigcirc \cdots \bigcirc \bigcirc_m \bigcirc_m \bigcirc.$$

If a clause c_j has x_i, then there is a directed edge from the left node of the pair $\bigcirc_i \bigcirc_i$ of the variable graph of x_i to c_j and a directed edge from c_j to the right node of the pair. If it is $\bar{x_i}$ then these two directed edges are reversed.

If there is a satisfying assignment of a 3CNF formula, then every variable x_i is either 1 or 0. If $x_i \leftarrow 1$, then the path starts from the left end of the *doubly linked graph* of x_i. If $x_i \leftarrow 0$, then the path starts from the right end of the *doubly linked graph* of x_i.

So there is a path from s through different variable nodes to d. To cover the nodes corresponding to the clauses, take one literal per clause that makes

[7] Actually we reduce SAT formula.

it *true*. Let the literal l_i (x_i or $\bar{x_i}$) is true for the clause c_j. Break the path of the *doubly linked graph* of x_i and include c_j in it.

In the other direction, if there is a Hamiltonian path from s to d, then there is a *truth value* assignment for the formula.

The number of vertices of the formula is $2 + m + (3m + 1) + (n - 1)$. So the encoding of the graph is a polynomial over the encoding of the formula.

QED.

1.5 Search Problem

We have asked membership question about the languages in **NP** e.g. whether the formula is satisfiable, whether the graph has an independent set of size k, whether the directed graph has a Hamiltonian path etc. These are decision problems.

We may search for solution, if there is one, for each such problems e.g. give a satisfying assignment of the formula, give an independent set of size k, give a Hamiltonian path etc.

Search problems are in general more difficult than the corresponding decision problem. It is easier to answer whether a positive integer (> 1) is composite, but more difficult to get its factorization. But if an **NP**-*complete* problems can be solved in polynomial time i.e. $\mathbf{P} = \mathbf{NP}$, then the certificate of any **NP** language can be generated in polynomial time.

Proposition 10. If $\mathbf{P} = \mathbf{NP}$, then for each $L \in \mathbf{NP}$ and its verifier V, there is a polynomial time Turing machine M that can generate a certificate w with respect to V, when run on $x \in L$.

Proof: We need to show that, if $\mathbf{P} = \mathbf{NP}$, then for each polynomial time bounded Turing machine M and for each polynomial $p(n)$, there is a polynomial time bounded Turing machine M' with the following property.

For every $x \in \{0,1\}^n$, if there is a $w \in \{0,1\}^{p(n)}$ such that M accepts $< x, w >$ i.e. $M(x, w) = 1$, then M' on input x produces w as the output i.e. $M'(x) = w$.

We consider the case of SAT. We assume that a Turing machine A decides the membership of SAT in polynomial time (this amounts to saying $\mathbf{P} = \mathbf{NP}$). We show that there is polynomial time Turing machine B, that on input of a satisfiable CNF formula ϕ of n variables, $\phi(x_1, \cdots, x_n)$, produces a satisfying assignment.

The Turing machine B works as follows:

1. Run A on ϕ to check whether it is satisfiable or not.
2. If ϕ is satisfiable, then for $i \leftarrow 1, \cdots, n$ do the following steps.
3. Assign x_i to 0, simplify the formula to $n - i$ variables., and run A to check whether $\phi(v_i, \cdots, v_{i-1}, 0, x_{i+1}, \cdots, x_n)$ is satisfiable, where $v_1, \cdots, v_{i-1}$ are already known assignments.
4. If it is, continue; otherwise continue with $\phi(v_i, \cdots, v_{i-1}, 1, x_{i+1}, \cdots, x_n)$ (simplified).
5. At the end either it is known that ϕ is unsatisfiable, or we have the satisfying assignment.

B is clearly polynomial time Turing machine.

Any $L \in \mathbf{NP}$ is *Levin reducible* to SAT, so a satisfying assignment of $f(x) = \phi_x \in$ SAT can be mapped back to the witness of $x \in L$. QED.

1.6 Reduction to SAT

The set of **NP**-complete problems is closed under Karp-reduction. An obvious question is how do we reduce $INDSET$ to SAT. This time the input is $< G, k >$, where $G = (V, E)$ is an undirected graph and k is a positive integer. The element $< G, k > \in INDSET$ if G has a independent set of size k. We define a computable map $f : \{0, 1\}^* \to \{0, 1\}^*$ such that $f(G, k) = \phi$ and

$$< G, k > \in INDSET \text{ if and only if } \phi \text{ is satisfiable.}$$

We need to choose the boolean variables and encode the independent set constraints as a boolean formula. This is in the similar line of encoding the computation of an NTM as a boolean formula.

Let $V = \{v_1, \cdots, v_n\}$ and an independent set of size k be $I = \{u_1, \cdots, u_k\}$. We introduce variables x_{ij}, where $1 \leq i \leq n$ and $1 \leq j \leq k$. The variable x_{ij} is *true* if $v_i = u_j$. Following is the set of constraints.

1. Each $u_j \in I$ must be some vertex of the graph. This is captured the following set of clauses.
$$\bigwedge_{j=1}^{k} \left(\bigvee_{i=1}^{n} x_{ij} \right).$$
2. No vertex should occure in I twice.
$$\bigwedge_{i=1}^{n} \bigwedge_{1 \leq j < m \leq k} (\neg x_{ij} \vee \neg x_{im}).$$
3. No element of I can be associated to two vertices of the graph.
$$\bigwedge_{j=1}^{k} \bigwedge_{1 \leq i < m \leq n} (\neg x_{ij} \vee \neg x_{mj}).$$
4. If two vertices are connected by an edge, then both of them cannot be in I.
$$\bigwedge_{1 \leq j < m \leq k} \bigwedge_{(v_i, v_l) \in E} (\neg x_{ij} \vee \neg x_{lm}).$$

This construction (reduction) can be done in time polynomial of the input length. The time complexity of the reduction is $O(nk + nk^2 + kn^2 + k^2 e) = O(k^2 n^2)$, where $e = |E|$

1.7 coNP, EXP, and NEXP

The class **coNP** was defined as follows:

$$\mathbf{coNP} = \{L \subseteq \{0, 1\}^* : \overline{L} \in \mathbf{NP}\}.$$

The class **P** is closed under complementation, so $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$. We already know that the following language are in **coNP**.

- Any language in **P** e.g. PRIME.
- $\overline{SAT} = \{\phi : \ \phi \text{ is unsatisfiable}\}$.
- $\overline{INDSET}$, $\overline{VERTEX-COVER}$, $\overline{CLIQUE}$ etc.

We may define the class **coNP** using a deterministic verifier.

Definition 5. A language $L \subseteq \{0,1\}^*$ is in **coNP** if and only if there is a polynomial $p : \mathbb{N}_0 \to \mathbb{N}_0$ and a polynomial time Turing machine so that for all $x \in \{0,1\}^*$,

$$x \in L \text{ if and only if } \forall w \in \{0,1\}^{p(|x|)},\ M \text{ accepts } < x, w >.$$

This is actually negation of the definition of **NP**.
Let $L \in$ **coNP**. So $\overline{L} \in$ **NP**. For all $x \in \{0,1\}^*$,

$$x \notin L \text{ if and only if } x \in \overline{L}.$$

There is a polynomial time bounded deterministic Turing machine V, a polynomial $p(n)$, and a witness $w \in \{0,1\}^{p(|x|)}$, such that V accepts $< x, w >$ if and only if $x \in \overline{L}$ i.e.

$$x \notin L \text{ if and only if } \exists w \in \{0,1\}^{p(|x|)},\ V \text{ accepts } < x, w > .$$

Equivalently,

$$x \in L \text{ if and only if } \neg(\exists w \in \{0,1\}^{p(|x|)},\ V \text{ accepts } < x, w >),$$

i.e.

$$x \in L \text{ if and only if } \forall w \in \{0,1\}^{p(|x|)},\ \overline{V} \text{ accepts } < x, w >,$$

where $\overline{V}$ is same as V in all respect, but the *accept* and *reject* states exchanged.

A language L is **coNP** complete if (i) it is in **coNP**, and (ii) every language L' in **coNP** is Karp reducible to L.

Proposition 11. Following language is **coNP**-*complete*.

$$TAUTOLOGY = \{\phi : \ \phi \text{ is a Boolean formula satisfiable by any assignment}\}.$$

Note that a formula $\phi \in$ TAUTOLOGY if and only if $\neg\phi$ is *unsatisfiable*.
Proof: If ϕ is a Boolean formula with n variables, then it is a *tautology* if and only if it is satisfied by any assignment of n variables. So there is a polynomial time Turing machine V such that for any $x \in \{0,1\}^n$, V will evaluate ϕ with x as assignment to its variables. The formula ϕ is a *tautology* if it evaluates to *true* (i.e. 1) for all x. So $TAUTOLOGY \in$ **coNP**.

We now show that every language $L \in$ **coNP** is Karp reducible to TAUTOLOGY. We take $\overline{L}$, the complement of L. If $L \in$ **coNP**, then $\overline{L} \in$ **NP**. So by Cook-Levin reduction we get ϕ_x. We know, $x \in L$ if and only if $x \notin \overline{L}$ if and only if ϕ_x is *unsatisfiable*. So, $x \in L$ if and only if $\neg\phi_x$ is a *tautology*.

The reduction is, for all $x \in \{0,1\}^*$, create ϕ_x by Cook-Levin reduction and take the negation of the formula. QED.

It is clear that if **P** = **NP**, then **NP** = **coNP** = **P**.

Definition 6. We define the class **NEXP** = $\bigcup_{c \geq 1}$ **NTIME**(2^{n^c}). By definition we have **P** $\subseteq$ **NP** $\subseteq$ **EXP** $\subseteq$ **NEXP**. We prove the following proposition.

Proposition 12. If $\mathbf{EXP} \neq \mathbf{NEXP}$, then $\mathbf{P} \neq \mathbf{NP}$.
Proof: We prove the contrapositive statement. We assume $\mathbf{P} = \mathbf{NP}$ and prove that $\mathbf{EXP} = \mathbf{NEXP}$.
Let $L \in \mathbf{NTIME}(2^{n^c})$. So a non-deterministic Turing machine N decides L in time 2^{n^c}. We define the language

$$L_{pad} = \left\{ < x, 1^{2^{|x|^c}} >: \ x \in L \right\},$$

and claim that $L_{pad} \in \mathbf{NP}$. The non-deterministic Turing machine N_{pad} for L_{pad} is as follows.
N_{pad}: input y

1. Nondeterministically it guesses a z, and computes $2^{|z|^c}$, so that $y =< z, 1^{2^{|z|^c}} >$. It *rejects* the input if no such z is found.
2. Otherwise, simulate N on z for $2^{|z|^c}$ steps.
3. If N accepts z, then *accept*, else *reject*.

The running time of N_{pad} is polynomial in $|y|$, so $L_{pad} \in \mathbf{NP}$. But according to our assumption $L_{pad} \in \mathbf{P}$. But then $z \in L$ if and only if $< z, 1^{2^{|x|^c}} > \in L_{pad}$. The padding string can be attached to z in exponential time and membership of $< z, 1^{2^{|x|^c}} > \in L_{pad}$ in L_{pad} can be tested in polynomial (on the length of $< x, 1^{2^{|x|^c}} >$) time.
Therefore the membership of x in L is determined in exponential (on the length of x). So $L \in \mathbf{EXP}$ i.e. $\mathbf{NEXP} \subseteq \mathbf{EXP}$. QED.

References

[MS] *Theory of Computation* by Michael Sipser, Pub. Cengage Learning, 2007, ISBN 978-81-315-0513-7.

[CHP] *Computational Complexity* by Christos H Papadimitriou, Pub. Addision-Wesley, 1994, ISBN 0-201-53082-1.

[DCK1] *Theory of Computation* by Dexter C Kozen, Pub. Springer, 2006, ISBN 978-81-8128-696-3.

[FCH] F C Hennie, *One-Tape, Off-Line Turing Machine Computations*, in Information and Control 8, pp 553-578, 1965.

[JH] J Hartmanis, *Computational Complexity of One-Tape Turing Machine Computation*, in JACM, vol. 15, No. 2, pp 325-339, April, 1968.

[LAL] L A Levin, *Universal search problems*, Problems of Information Transmission, 9 (3): 115116 (Russian), translated into English by Trakhtenbrot, B. A. (1984). "A survey of Russian approaches to perebor (brute-force searches) algorithms". Annals of the History of Computing 6 (4): 384400.

[SABB] *Computational Complexity, A Modern Approach* by Sanjeev Arora & Boaz Barak, Pub. Cambridge University Press, 2009, ISBN 978-0-521-42426-4.

[SAC] S A Cook, *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151158, 1971.

[RMK] R M Karp, *Reducibility Among Combinatorial Problems*, in R E Miller and J W Thatcher, ed. *Complexity of Computer Communications*, pp 85-103, Plenum, 1972.

[NPMJF] Nicholas Pippenger, And Michael J Fischer, *Relation Among Complexity Measurse*, in JACM 26, 2 (April 1979), 361-381.