Theory of Computing
Time Complexity

Ming-Hsien Tsai

Department of Information Management
National Taiwan University

Spring 2019

(original created by Bow-Yaw Wang)

Time for Deciding a Language

- Let us consider $A = \{0^n 1^n : n \geq 0\}$.
- How much time does a single-tape TM need to decide A?
- Consider
  M_1 = "On input string w:
  1. Scan the tape and reject if a `0` appears after a *1*.
  2. Repeat if `0` or `1` appear on the tape:
     1. Scan across the tape, cross a `0` and a `1`.
  3. If `0`'s or `1`'s still remain, reject. Otherwise, accept."
- How much "time" does M_1 need for an input w?

Time Complexity

Definition 1

Let M be a TM that halts on all inputs. The running time (or time complexity) of M is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the running time of M on any input of length n.

- If $f(n)$ is the running time of M, we say M runs in time $f(n)$ and M is an $f(n)$ time TM.
- In worst-case analysis, the longest running time of all inputs of a particular length is considered.
- In average-case analysis, the average of all running time of inputs of a particular length is considered instead.
- We only consider worst-case analysis in the course.

Big-O and Small-O

Definition 2

Let $f, g : \mathbb{N} \to \mathbb{R}^+$. $f(n) = O(g(n))$ if there are $c, n_0 \in \mathbb{Z}^+$ such that for all $n \geq n_0$,

$$f(n) \leq c(g(n)).$$

- $g(n)$ is an upper bound (or an asymptotic upper bound) for $f(n)$.
- $n^c (c \in \mathbb{R}^+)$ is a polynomial bound.
- $2^{n^d} (d \in \mathbb{R}^+)$ is an exponential bound.

Definition 3

Let $f, g : \mathbb{N} \to \mathbb{R}^+$. f (n) = o (g (n)) if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

That is, for any $c \in \mathbb{R}$, there is an n_0 that $f(n) < c(g)$ for all $n \geq n_0$.

Time Complexity of M_1

- Recall
  M_1 = "On input string w:
  1. Scan the tape and reject if a `0` appears after a *1*.
  2. Repeat if `0` or `1` appear on the tape:
     1. Scan across the tape, cross a `0` and a `1`.
  3. If `0`'s or `1`'s still remain, reject. Otherwise, accept."
- Let $|w| = n$.
  - Step 1 takes $O(n)$ (precisely, $\leq n$).
  - Step 2 has $O(n)$ iterations (precisely, $\leq n/2$).
    - ★ An iteration takes $O(n)$ (precisely, $\leq n$).
  - Step 3 takes $O(n)$ (precisely, $\leq n$).
- The TM M_1 decides $A = \{0^n 1^n : n \geq 0\}$ in time $O(n^2)$.
  - $O(n^2) = O(n) + O(n) \times O(n) + O(n)$.

Time Complexity Class

Definition 4

Let $t : \mathbb{N} \to \mathbb{R}^+$. The time complexity class $TIME(t(n))$ is the collection of all languages that are decided by a $O(t(n))$ time TM.

- $A = \{0^n 1^n : n \geq 0\}$ is decided by M_1 in time $O(n^2)$. $A \in TIME(n^2)$.
- Time complexity classes characterizes languages, not TM's.
  - We don't say $M_1 \in TIME(n^2)$.
- A language may be decided by several TM's.
- Can A be decided more quickly asymptotically?

Models and Time Complexity

- Consider the following TM:
  M_2 = "On input string w:
  1. Scan the tape and reject if a `0` appears after a `1`.
  2. Repeat if `0` or `1` appear on the tape:
     1. Scan the tape and check if the total number of `0`'s and `1`'s is even. If not, reject.
     2. Scan the tape, cross every other `0` from the first `0`, and cross every other `1` from the first `1`.
  3. If `0`'s or `1`'s still remain, reject. Otherwise, accept."
- Analysis of M_2.
  - Step 1 takes $O(n)$.
  - Step 2 has $O(\lg n)(= \log_2(n))$ iterations (why?). At each iteration,
    - ★ Step 1 takes $O(n)$.
    - ★ Step 2 takes $O(n)$.
  - Step 3 takes $O(n)$.
- M_2 decides A in time $O(n \lg n)$.
  - $O(n \lg n) = O(n) + O(\lg n) \times O(n) + O(n)$.

Models and Time Complexity

- Consider the following two-tape TM:
  M_3 = "On input string w:
  1. Scan tape 1 and reject if a `0` appears after a `1`.
  2. Scan tape 1 and copy the `0`'s onto tape 2.
  3. Scan tape 1 and cross a `0` on tape 2 for a `1` on tape 1.
  4. If all `0`'s are crossed off before reading all `1`'s, reject. If some `0`'s are left after reading all `1`'s, reject. Otherwise, accept."
- Analysis of M_3.
  - Each step takes $O(n)$.
- For the same language $A = \{\mathtt{0}^n\mathtt{1}^n : n \geq 0\}$.
  - The TM M_1 decides A in time $O(n^2)$, the TM M_2 decides A in time $O(n \lg n)$, and the two-tape M_3 decides A in time $O(n)$.
- In computability theory, all reasonable variants of TM's decide the same language (Church-Turing thesis).
- In complexity theory, different variants of TM's may decide the same in different time.

Complexity Relationship with Multitape TM's

Theorem 5

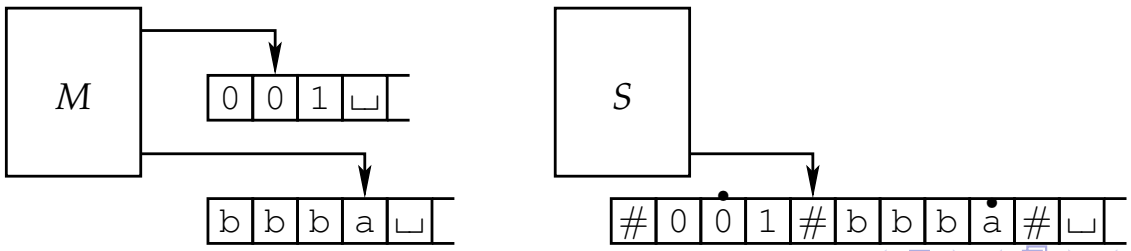
Let $t(n)$ be a function with $t(n) \geq n$. Every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape TM.

Proof.

We analyze the simulation of a k-tape TM M is by the TM S. Observe that each tape of M has length at most $t(n)$ (why?).
For each step of M, S has two passes:

- The first pass gathers information ($O(kt(n))$).
- The second pass updates information with at most k shifts ($O(k^2t(n))$).

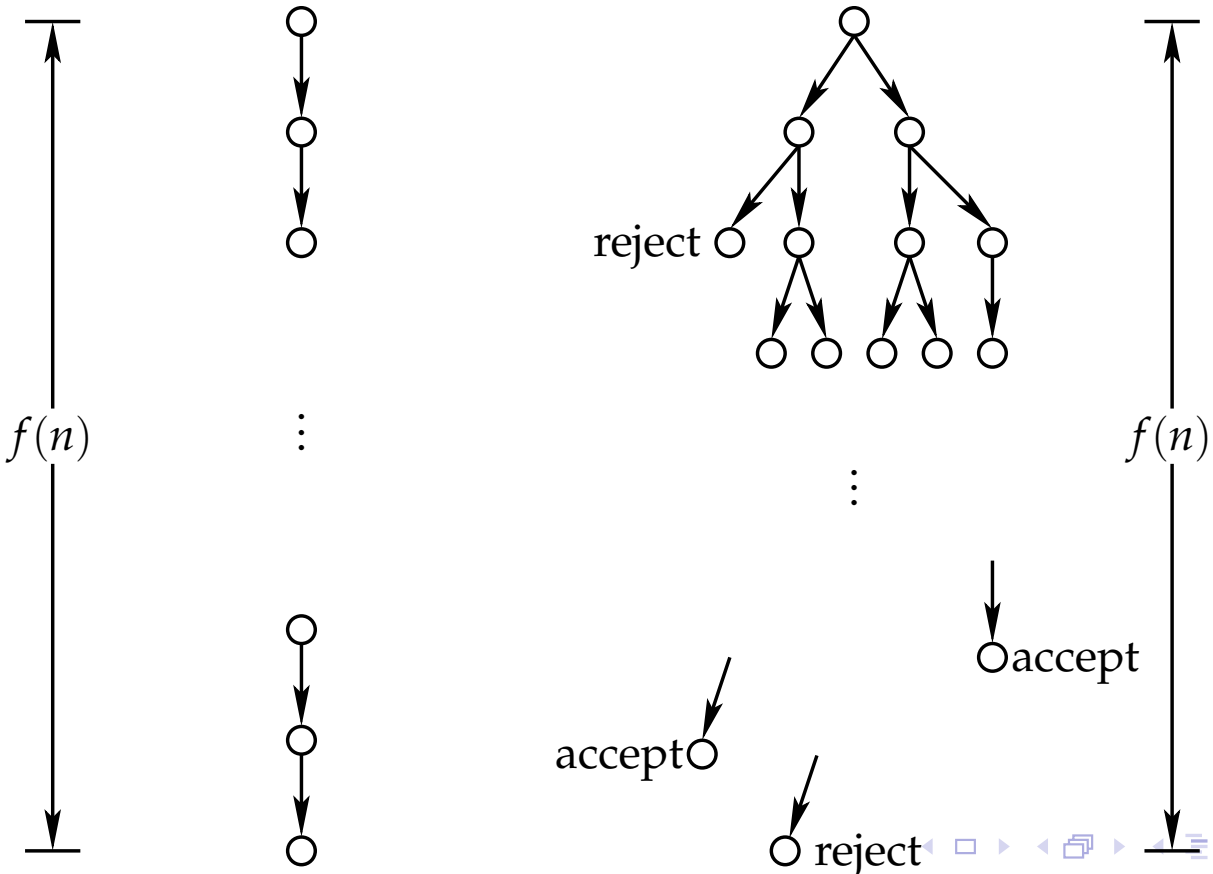
Hence S takes $O(n) + O(k^2t^2(n))$ $(= O(n) + O(t(n)) \times O(k^2t(n)))$. Since $t(n) \geq n$, we have S runs in time $O(t^2(n))$ (k is independent of the input). □



Time Complexity of Nondterministic TM's

Definition 6

Let N be a nondeterministic TM that is a decider. The running time of N is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps among any branch of N's computation on input of length n.



Complexity Relationship with NTM's

Theorem 7

Let $t(n)$ be a function with $t(n) \geq n$. Every $t(n)$ time single-tape NTM has an equivalent $2^{O(t(n))}$ time single-tape TM.

Proof.

Let N be an NTM running in time $t(n)$. Recall the simulation of N by a 3-tape TM D with the address tape alphabet $\Sigma_b = \{1, 2, \ldots, b\}$ (b is the maximal number of choices allowed in N).
Since N runs in time $t(n)$, the computation tree of N has $O(b^{t(n)})$ nodes. For each node, D simulates it from the start configuration and thus takes time $O(t(n))$. Hence the simulation of N on the 3-tape D takes $2^{O(t(n))} (= O(t(n)) \times O(b^{t(n)}))$ time.
By Theorem 5, D can be simulated by a single-tape TM in time $(2^{O(t(n))})^2 = 2^{O(t(n))}$. □

The Class P

- It turns out that reasonable deterministic variants of TM's can be simulated by a TM with a polynomial time overhead.
  - multitape TM's, TM's with random access memory, etc.
- The polynomial time complexity class is rather robust.
  - That is, it remains the same with different computational models.

Definition 8

P is the class of languages decidable in polynomial time on a determinsitic single-tape TM. That is,

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

- We are interested in intrinsic characters of computation and hence ignore the difference among variants of TM's in this course.
- Solving a problem in time $O(n)$ and $O(n^{100})$ certainly makes lots of difference in practice.

Problems in P

- We will give some problems and their deciders in P.
- We will use a reasonable encoding for our problems.
- For natural numbers, we represent them in base $k \geq 2$ notation.
  - Uniary encoding is not reasonable.
  - $17 = 10001_2 = 11111111111111111_1$.
- For graphs, we use adjacency matrices.
  - The (i, j)th entry is 1 if there is an edge from node i to node j.
- Note that the size of a graph is a polynomial in our reasonable encoding of the graph.
- Consider the following problem:

$$PATH = \{\langle G, s, t\rangle : \text{ there is a path from } s \text{ to } t \text{ in the directed graph } G\}.$$

PATH is in *P*

Theorem 9

$PATH \in P$.

Proof.

Consider
$M =$ "On input $\langle G, s, t \rangle$ where G is a directed graph with nodes s and t:

1. Place a mark on s.
2. Repeat until no more nodes are marked:
    1. If there is an edge (a, b) from a marked node a to an unmarked node b, mark b.
3. If t is marked, accept. Otherwise, reject."

Let m be the number of nodes in G. Step 1 takes $O(1)$. Step 2 has $O(m)$ iterations; each iteration takes $O(m^2)$. Step 3 take $O(1)$. Hence M is a polynomial time algorithm for *PATH*. □

RELPRIME is in P

- Consider $RELPRIME = \{\langle x, y\rangle : x \text{ and } y \text{ are relatively prime}\}$.

Theorem 10

$RELPRIME \in P$.

Proof.

Recall the Euclidean algorithm:
$E =$ "On input $\langle x, y\rangle$ where $x, y \in \mathbb{Z}^+$ in binary:

1. Repeat until $y = 0$
   1. $x \leftarrow x \mod y$.
   2. Exchange x and y.
2. Output x".

Consider
$R =$ "On input $\langle x, y\rangle$ where $x, y \in \mathbb{Z}^+$ in binary:

1. Run E on $\langle x, y\rangle$.
2. If E outputs 1, accept; otherwise, reject."

RELPRIME is in P

Proof (cont'd).

It remains to analyze the time complexity of E. We show each execution (except perhaps the first) of Step 1.1 cuts the value of x by a half.
Observe that $x > y$ for each execution (except perhaps the first). Right before Step 1.1,

- If $\frac{x}{2} \geq y$, then $x \mod y < y \leq \frac{x}{2}$.
- If $\frac{x}{2} < y$, then $x \mod y = x - y \leq \frac{x}{2}$.

Hence x draps by a half right after Step 1.1. Since x and y exchanges values at Step 1.2, x and y drop by a half every other iteration. Hence Step 1 has $O(\lg x + \lg y)$ iterations. Recall we are use the binary encoding. Step 1 has a polynomial number of iterations. Moreover, comparison, modulo, and assignments take a polynomial number of time. R decides *RELPRIME* in polynomial time. □

Context-Free Languages are in P

Theorem 11

Every context-free language is a member of P.

Proof.

Consider $D =$ "On input $w = w_1 w_2 \cdots w_n$: // G is in Chomsky normal form

1. If $w = \epsilon$ and $S \longrightarrow \epsilon \in G$, accept.
2. For $i = 1$ to n do // Handle substrings of length 1
   For each variable A
   If $A \longrightarrow b \in G$ and $w_i = b$, put A in $table(i, i)$.
3. For $l = 2$ to n do // Handle substrings of length l
   1. For $i = 1$ to $n - l + 1$ do // Handle the substring at $w_i w_{i+1} \cdots w_j$
      1. $j = i + l - 1$.
      2. For $k = i$ to $j - 1$ do // Consider $w_i \cdots w_k$ and $w_{k+1} \cdots w_j$
         For each $A \longrightarrow BC \in G$ do
         If $B \in table(i, k)$ and $C \in table(k + 1, j)$, add A to $table(i, j)$.
4. If $S \in table(1, n)$, accept; otherwise, reject."

Context-Free Langauges are in P

Proof.

Let G have v variables and r rules (v, r are independent of $|w|$).

- Step 1 executes once.
- Step 2 has $O(n)$ iterations.
  - Each iteration has $O(v)$ subiterations.
- Step 3 has $O(n)$ iterations.
  - Each iteration has $O(n)$ subiterations.
    - Each subiteration has $O(n)$ subsubiterations; each subsubiterations has r subsubsubiterations.
- Step 4 executes once.

Since table lookup and insertion take a polynomial number of time, D runs in polynomial time. □

The Class NP

Definition 12

A verifier for a language A is an algorithm V where

$$A = \{w : V \text{ accepts } \langle w, c \rangle \text{ for some } c\}.$$

c is a certificate or proof of membership in A. A polynomial time verifier runs in polynomial time in the length of w (not $\langle w, c \rangle$). A language A is polynomially verifiable if it has a polynomial time verifier.

- Note that a certificate has a length polynomial in $|w|$.
  - Otherwise, V cannot run in polynomial time in $|w|$.

Definition 13

NP is the class of languages that have polynomial time verifiers.

Hamiltonian Paths

- A Hamiltonian path in a directed graph G is a path that goes through every node exactly once. Consider

$$HAMPATH = \{\langle G, s, t\rangle : \ G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}.$$

- $HAMPATH \in NP$.
  - Verifying whether c is a Hamiltonian path from s to t can be done in polynomial time.
  - A certificate for $\langle G, s, t\rangle \in HAMPATH$ is a Hamiltonian path from s to t.
- Finding a Hamiltonian path from s to t seems harder.

Composites

- A natural number is composite if it is the product of two integers greater than 1. Consider

$$COMPOSITES = \{\langle x \rangle : x = pq \text{ for } p, q \in \mathbb{Z}^+ \text{ and } p, q > 1\}.$$

- $COMPOSITES \in NP$.
  - Verifying whether x is the product of two integers p, q with $p, q > 1$ can be done in polynomial time.
  - A certificate for $\langle x \rangle \in COMPOSITES$ is $\langle p, q \rangle$ with $p, q \in \mathbb{Z}^+$, $p, q > 1$, and $x = pq$.
- Finding $p, q \in \mathbb{Z}^+$ with $p, q \in \mathbb{Z}^+$, $p, q > 1$, and $x = pq$ seems harder.
  - Can we find p in $\{2, 3, \ldots, \lceil \sqrt{x} \rceil\}$? Is it a polynomial time algorithm?
- There is however a polynomial time algorithm for $COMPOSITES$.

An NTM Dedicing *HAMPATH*

- Consider
  $N_1 =$
  "On input $\langle G, s, t \rangle$ where G is a directed graph with nodes s, t:
  1. Nondeterministically write m numbers $p_1, \ldots, p_m$ (G has m nodes).
  2. If there is any repetition, reject.
  3. If $s \neq p_1$ or $t \neq p_m$, reject.
  4. If (p_i, p_{i+1}) is not an edge of G for some $1 \leq i < m$, reject.
  5. Otherwise, accept."
- Since N_1 runs in polynomial time and $L(N_1) = HAMPATH$, N_1 decides *HAMPATH*.

NP and NTM's

Theorem 14

A language is in NP if and only if it is decided by a nondeterministic polynomial time Turing machine.

Proof.

Let V be a verifier for a language A running in time n^k. Consider
$N =$ "On input w of length n:

1. Nondeterministically select string c of length $\leq n^k$.
2. Run V on $\langle w, c \rangle$.
3. If V accepts, accept; otherwise, reject."

Conversely, let the NTM N decide A and c the address of an accepting configuration in the computation tree of N. Consider
$V =$ "On input $\langle w, c \rangle$:

1. Simulate N on w from the start configuration by c.
2. If the configuration with address c is accepting, accept; otherwise, reject." $\square$

The Nondeterministic Time Complexity Class

Definition 15

NTIME (t (n)) = { L : L is a language decided by a O (t (n)) time NTM }.

Corollary 16

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

- Recall that class $TIME(t(n))$ and

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

Cliques in Graphs

- A clique in an undirected graph is subgraph where every two nodes are connected.
- A k-clique is a clique with k nodes.
- Consider

$$CLIQUE = \{\langle G, k\rangle : G \text{ is an undirected graph with a } k\text{-clique}\}.$$

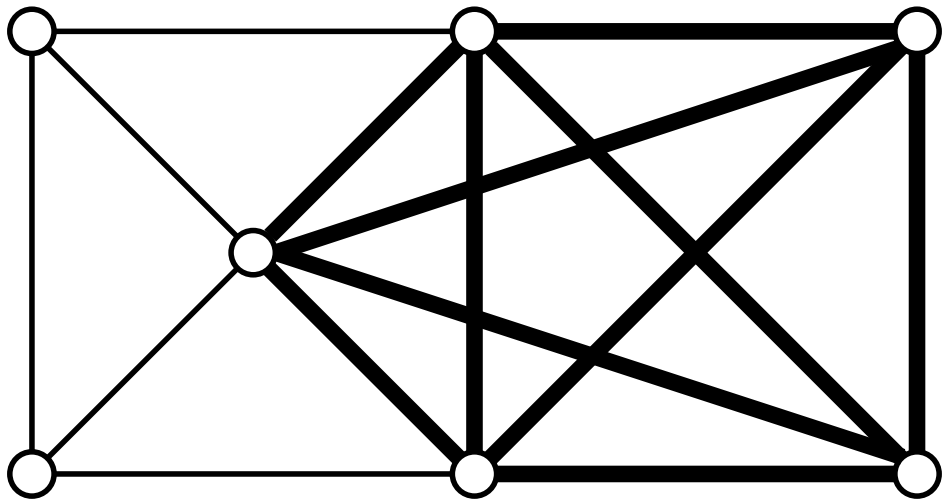


Figure: A Graph has a 5-Clique

CLIQUE is in *NP*

Theorem 17

CLIQUE $\in$ *NP*.

Proof.

Consider

$V =$ "On input $\langle\langle G, k\rangle, c\rangle$:

1. Check if c is a set of k nodes in G.
2. Check if G contains all edges connecting nodes in c.
3. If both pass, accept; otherwise, reject."

Alternatively, consider

$N =$ "On input $\langle G, k\rangle$ where G is an undirected graph:

1. Nondeterministically select a set c of k nodes of G.
2. Check if G contains all edges connecting nodes in c.
3. If yes, accept; otherwise, reject." □

Subset Sums

- Consider

$$SUBSETSUM = \{\langle S, t\rangle : \quad S = \{x_1, \ldots, x_k\} \text{ a multiset and for some } \{y_1, \ldots, y_l\} \subseteq S, \Sigma y_i = t\}$$

Theorem 18

$SUBSETSUM \in NP$.

Proof.

$V =$ "On input $\langle\langle S, t\rangle, c\rangle$:

1. Check if S contains all the numbers in c.
2. Check if c sums to t.
3. If both pass, accept; otherwise, reject."

Alternatively, $N =$ "On input $\langle S, t\rangle$ where S is a multiset:

1. Nondeterministcally select a subset c of numbers from S.
2. Check if c sums to t.
3. If both pass, accept; otherwise, reject." □

The Class *coNP*

Definition 19

$coNP = \{L : \overline{L} \in NP\}$.

- $\overline{HAMPATH} \in coNP$ since $\overline{\overline{HAMPATH}} = HAMPATH \in NP$.
  - $\overline{HAMPATH}$ does not appear to be polynomial time verifiable.
  - What is a certificate showing there is no Hamiltonian path?
- $\overline{CLIQUE} \in coNP$.
  - $\overline{CLIQUE}$ does not appear to be polynomial time verifiable.
  - What is a certificate showing there is no k-clique?
- $\overline{SUBSETSUM} \in coNP$.
  - $\overline{SUBSETSUM}$ does not appear to be polynomial time verifiable.
  - What is a certificate showing there is no subset summing to t?
- We do not know if *coNP* is different from *NP*.

P vs NP

- Recall
  - P is the class of languages which membership can be decided quickly.
  - NP is the class of languages which membership can be verified quickly.
- We have shown $HAMPATH, CLIQUE, SUBSETSUM \in NP$.
- We have shown $PATH, RELPRIME \in P$.
- Since a polynomial time TM is also a polynomial NTM, we have

$$L \in P \text{ implies } L \in NP \text{ for every language } L.$$

P vs NP

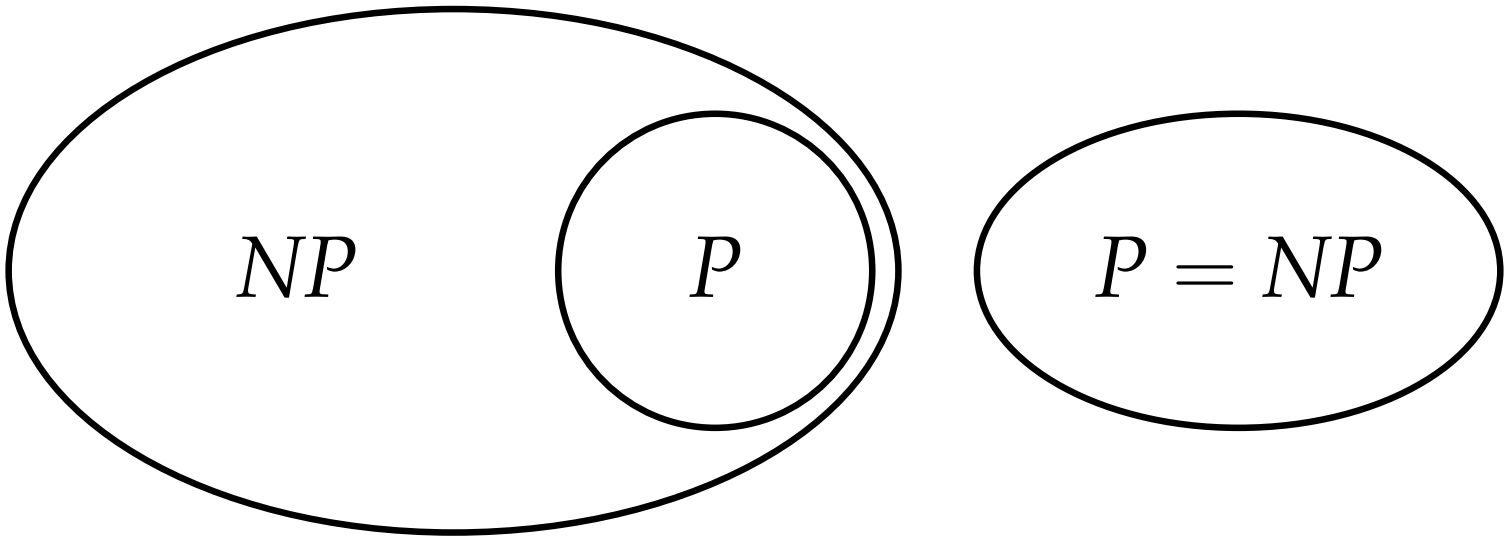


Figure: Possible Relation between P and NP

- To the best of our knowledge, we only know

$$NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k}). \qquad \text{(Theorem 7)}$$

- Particularly, we do no know if $P \stackrel{?}{=} NP$.

Satisfiability

- Let $\mathbb{B} = \{0, 1\}$ be the truth values.
- A Boolean variable takes values from $\mathbb{B}$.
- Recall the Boolean operations

$$\begin{array}{lllllllll} 0 \wedge 0 & = & 0 & 0 \vee 0 & = & 0 & & & \\ 0 \wedge 1 & = & 0 & 0 \vee 1 & = & 1 & \overline{0} & = & 1 \\ 1 \wedge 0 & = & 0 & 1 \vee 0 & = & 1 & \overline{1} & = & 0 \\ 1 \wedge 1 & = & 1 & 1 \vee 1 & = & 1 & & & \end{array}$$

- A Boolean formula is an expression constructed from Boolean variables and opearations.
  - $\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ is a Boolean formula.
- A Boolean formula is satisfiable if an assignments of 0's and 1's to Boolean variables makes the formula evaluate to 1.
  - ϕ is satisfiable by taking $\{x \mapsto 0, y \mapsto 1, z \mapsto 0\}$.

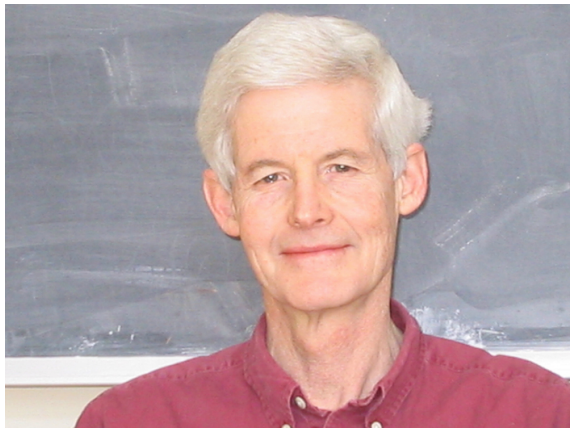
The Satisfiability Problem

- The satisfiability problem is to test whether a Boolean formula is satisfiable.
- Consider

$$SAT = \{\langle \phi \rangle : \phi \text{ is a satisfiable Boolean formula}\}.$$

Theorem 20 (Cook-Levin)

$SAT \in P$ if and only if $P = NP$.



Polynomial Time Reducibility

Definition 21

$f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if a polynomial time TM M halts with only $f(w)$ on its tape upon any input w.

Definition 22

A language A is polynomial time mapping reducible (polynomial time reducible, or polynomial time many-one reducible) to a language B (written $A \leq_P B$) if there is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ that

$$w \in A \text{ if and only if } f(w) \in B \text{ for every } w.$$

f is called the polynomial time reduction of A to B.

- Recall the definitions of computable functions and mapping reducibility.

Properties about Polynomial Time Reducibility

Theorem 23

If $A \leq_P B$ and $B \in P$, $A \in P$.

Proof.

Let the TM M decide B and f a polynomial time reduction of A to B. Consider

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on $f(w)$."

Since the composition of two polynomials is again a polynomial, N runs in polynomial time. □

The 3*SAT* Problem

- A literal is a Boolean variable or its negation.
- A clause is a disjunction ($\vee$) of literals.
  - $x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4$ is a clause.
- A Boolean formula is in conjunctive normal form (or a CNF-formula) if it is a conjunction ($\wedge$) of clauses.
  - $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee x_2 \vee \overline{x_5}) \wedge (x_4 \vee x_6)$ is a CNF-formula.
- In a satisfiable CNF-formula, each clause must contain at least one literal assigned to 1.
- A Boolean formula is a 3CNF-formula if it is a CNF-formula whose clauses have three literals.
  - $(x_1 \vee \overline{x_3} \vee x_4) \wedge (x_2 \vee x_2 \vee \overline{x_5}) \wedge (x_4 \vee x_5 \vee \overline{x_6})$ is a 3CNF-formula.
- Consider

$$3SAT = \{\langle \phi \rangle : \phi \text{ is a satisfiable 3CNF-formula}\}.$$

$3SAT \leq_P CLIQUE$

Theorem 24

$3SAT \leq_P CLIQUE$.

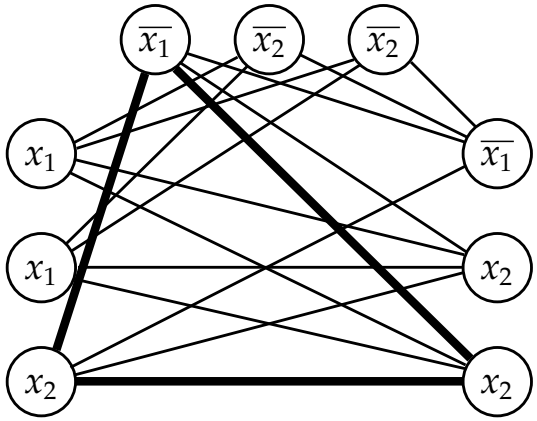
Proof.

Given a 3CNF-formula $\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$, we would like to find a graph G and a number k such that $\langle \phi \rangle \in 3SAT$ if and only if $\langle G, k \rangle \in CLIQUE$. We need gadgets to simulate Boolean variables and clauses in ϕ.

- For each clause $a_i \vee b_i \vee c_i$, add three corresponding nodes to G.
  - G has $3k$ nodes.
- For each pair of nodes in G, add an edge except when
  - the pair of nodes correspond to literals in a clause.
  - the pair of nodes correspond to complementary literals.

We next show that ϕ is satisfiable if and only if G has a k-clique.

$3SAT \leq_P CLIQUE$



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

Proof.

Suppose ϕ has a satisfying assignment. Each clause has at least one literal assigned to 1. We pick a node corresponding to true literal from each clause. Any pair of the chosen nodes do not belong to the same clause. Since a literal and its complement cannot be 1 simultaneously, any pair of the chosen nodes are not complementary. Hence there is an edge between any pair of the chosen nodes. We have a k-clique.
Conversely, suppose there is a k-clique. Since there is no edge between any two nodes in a clause, the k-clique must have one node from each of the k clauses. Moreover, there is no edge between complementary literals. Either a literal or its complement appears in the k-clique but not both. ϕ is satisfied by the assignment making literals in the clique true.
It is easy to see that G can be constructed from ϕ in polynomial time. □

NP-Completeness

Definition 25

A language B is *NP*-complete if

- B is in *NP*; and
- every A in *NP* is polynomial time reducible to B.

Theorem 26

If B is NP-complete and $B \in P$, then $P = NP$.

Theorem 27

If $C \in NP$, B is NP-complete, and $B \leq_P C$, then C is NP-complete.

Proof.

Since B is *NP*-complete, there is a polynomial time reduction f of A to B for any $A \in NP$. Since $B \leq_P C$, there is a polynomial time reduction g of B to C. $g \circ f$ is a polynomial time reduction of A to C. □

Cook-Levin Theorem

Theorem 28

SAT is NP-complete.

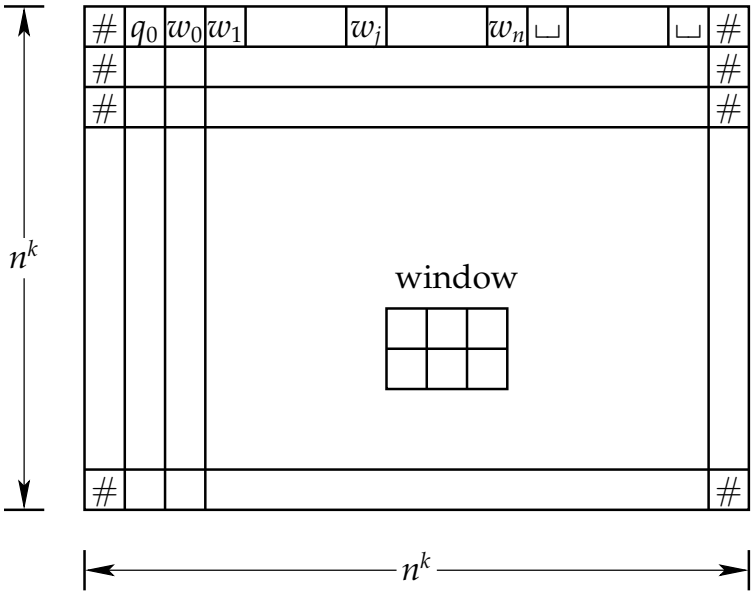
Proof.

For any Boolean formula ϕ, an NTM nondeterministically choose a truth assignment. It checks whether the assignment satisfies ϕ. If so, accept; otherwise, reject. Hence $SAT \in NP$.

Let $A \in NP$ and the NTM N decide A in n^k time. For any input w, a tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations along a branch of the computation of N on w. A tableau of size $n^k \times n^k$ has $n^k \times n^k$ cells. We assume each configuration starts and ends with a # symbol. A tableau is accepting if any of its rows is an accepting configuration.

Each accepting tableau for N on w corresponds to an accepting computation of N on w. We therefore construct a Boolean formula ϕ such that ϕ is satisfiable if and only if there is an accepting tableau for N on w.

Cook-Levin Theorem



Proof (cont'd).

Let $C = Q \cup \Gamma \cup \{\#\}$ where Q and Γ are the states and the tape alphabet of N. For $1 \le i, j \le n^k$ and $s \in C$, the Boolean variable $x_{i,j,s}$ denotes the content of the cell $cell[i, j]$. That is, $x_{i,j,s}$ is 1 if and only if $cell[i, j] = s$. To force each cell to contain exactly one symbol from C, consider

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in C, s \ne t} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right].$$

Cook-Levin Theorem

Proof (cont'd).

To force the tableau to begin with the start configuration, consider

$$\begin{aligned}\phi_{\text{start}} \quad = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}.\end{aligned}$$

To force an accepting configuration to appear in the tableau, consider

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}.$$

To force the configuration at row i yields the configuration at row $i+1$, consider a window of 2×3 cells. For example, assume $\delta(q_1, \mathtt{a}) = \{(q_1, \mathtt{b}, R)\}$ and $\delta(q_1, \mathtt{b}) = \{(q_2, \mathtt{c}, L), (q_2, \mathtt{a}, R)\}$. The following windows are valid:

a	q_1	b	a	q_1	b	a	a	q_1	#	b	a	a	b	a	b	b	b
q_2	a	c	a	a	q_2	a	a	b	#	b	a	a	b	q_2	c	b	b

Cook-Levin Theorem

Proof.

Since C is finite, there are only a finite number of valid windows. For any window W

c_1	c_2	c_3
c_4	c_5	c_6

, consider

$$\psi_W = x_{i,j-1,c_1} \wedge x_{i,j,c_2} \wedge x_{i,j+1,c_3} \wedge x_{i+1,j-1,c_4} \wedge x_{i+1,j,c_5} \wedge x_{i+1,j+1,c_6}$$

To force every window in the tableau to be valid, consider

$$\phi_{\text{move}} = \bigwedge_{1 \le i \le n^k, 1 \le j < n^k} \left(\bigvee_{W \text{ is a valid}} \psi_W \right).$$

Finally, consider the following Boolean formula:

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

$|\phi_{\text{cell}}| = O(n^{2k})$, $|\phi_{\text{start}}| = O(n^k)$, $|\phi_{\text{accept}}| = O(n^{2k})$, and $|\phi_{\text{move}}| = O(n^{2k})$. Hence $|\phi| = O(n^{2k})$. Moreover, ϕ can be constructed from N in time polynomial in n. □

3*SAT* is *NP*-Complete

Corollary 29

3SAT is NP-complete.

Proof.

We convert the Boolean formula ϕ in the proof of Theorem 28 into a 3CNF-formula.
We begin by converting ϕ into a CNF-formula.
Observe that the conjunction of CNF-formulae is again a CNF-formula. Note that ϕ_{cell}, ϕ_{start}, and ϕ_{accept} are already in CNF (why?). ϕ_{move} is of the following form:

$$\bigwedge_{1\le i\le n^k, 1\le j<n^k} \left(\bigvee_{W \text{ is valid}} (l_1 \wedge l_2 \wedge l_3 \wedge l_4 \wedge l_5 \wedge l_6) \right)$$

By the law of distribution, ϕ_{move} can be converted into a CNF-formula. Note that the conversion may increase the size of ϕ_{move}. Yet the size is independent of $|w|$. Hence the size of the CNF-formula ϕ still polynomial in $|w|$.
To a clause of k literals into clauses of 3 literals, consider $l_1 \mapsto (l_1 \vee l_1 \vee l_1)$,
$l_1 \vee l_2 \mapsto (l_1 \vee l_2 \vee l_2)$, and
$l_1 \vee l_2 \vee \cdots l_p \mapsto (l_1 \vee l_2 \vee z_1) \wedge (\overline{z_1} \vee l_3 \vee z_2) \wedge \cdots \wedge (\overline{z_{p-3}} \vee l_{p-1} \vee l_p)$. □

More *NP*-Complete Problems

- To find more *NP*-complete problems, we apply Theorem 27.
- Concretely, to show C is *NP*-complete, do
  - prove C is in *NP*; and
  - find a polynomial time reduction of an *NP*-complete problem (say, $3SAT$) to C.
- In Theorem 24, we have shown $3SAT \leq_P CLIQUE$. Therefore

Corollary 30

CLIQUE is NP-complete.

$\neq SAT$

- Let ϕ be a 3CNF-formula.
  - For instance $(x_0 \vee \overline{x}_1 \vee x_2) \wedge (\overline{x}_0 \vee x_1 \vee \overline{x}_3)$.
- An $\neq$-assignment to the variables of ϕ makes every clause of ϕ to have two literals with different truth values.
  - $\{x_0 \mapsto 1, x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 0\}$ is an $\neq$-assignment.
  - $\{x_0 \mapsto 1, x_1 \mapsto 0, x_2 \mapsto 1, x_3 \mapsto 0\}$ is not an $\neq$-assignment.
- Observe that the negation of an $\neq$-assignment is again an $\neq$-assignment.
  - $\{x_0 \mapsto 0, x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$ is an $\neq$-assignment.
- Define

$$\underline{\neq SAT} = \{\langle \phi \rangle : \phi \text{ is a 3CNF formula with an } \neq\text{-assignment}\}.$$

$\neq$ *SAT* is *NP*-Complete

Theorem 31

$\neq$ SAT is NP-complete.

Proof.

We reduce $3SAT$ to $\neq SAT$. Let b be a Boolean variable not in ϕ. Replace the i-th clause $(l_{i0} \vee l_{i1} \vee l_{i2})$ in ϕ with $(l_{i0} \vee l_{i1} \vee z_i)$ and $(\bar{z}_i \vee l_{i2} \vee b)$.

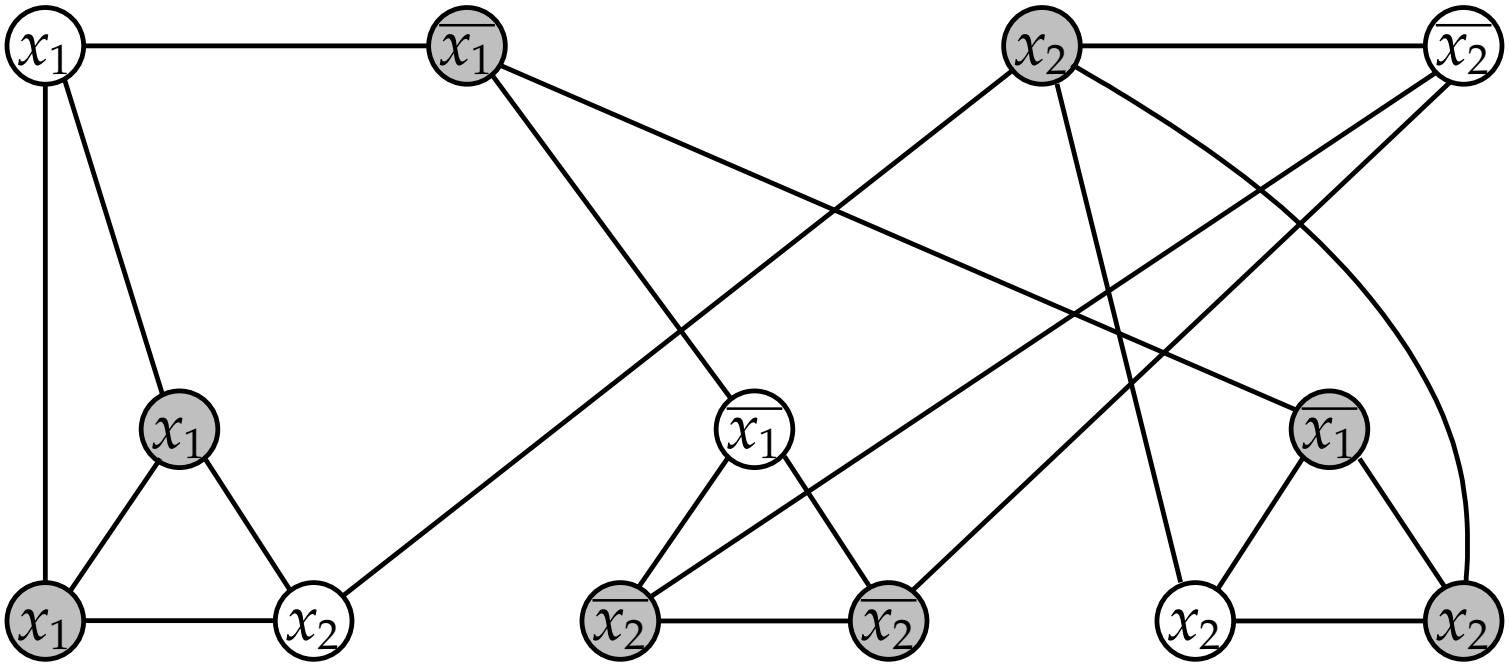
Assume ϕ is satisfiable. We extend the satisfying assignment by assigning b to 0 and z_i to $\begin{cases} 0 & \text{if } l_{i2} = 0 \text{ or } l_{i0} = l_{i1} = l_{i2} = 1 \\ 1 & \text{otherwise} \end{cases}$.

Conversely, assume we have an $\neq$-assignment. We show the $\neq$-assignment satisfies ϕ as well. Without loss of generality, assume $b = 0$ in the $\neq$-assignment. If $l_{i2} = 1$ in the $\neq$-assignment, the i-th clause in ϕ is satisfied. Otherwise, $\bar{z}_i = 1$ for $\bar{z}_i \vee l_{i2} \vee b$ has two literals of different values. Then $l_{i0} = 1$ or $l_{i1} = 1$ because $l_{i0} \vee l_{i1} \vee z_i$ has two literals of different values. The i-th clause of ϕ is satisfied as well. $\square$

Vertex Cover

- Let G be an undirected graph. A vertex cover of G is a set of nodes where every edge of G touches one of these nodes.
- Consider

$$VERTEXCOVER = \{\langle G, k\rangle : \quad G \text{ is an undirected graph with a } k\text{-node vertex cover}\}.$$



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

VERTEXCOVER is *NP*-Complete

Theorem 32

VERTEXCOVER is NP-complete.

Proof.

We show $3SAT \leq_P VERTEXCOVER$. Let ϕ be a 3CNF-formula with l variables and k clauses. We need gadgets to simulate variables and clauses in ϕ. We want to find a graph G such that ϕ is satisfiable if and only if G has a $l + 2k$-node vertex cover.

- For each variable x, add two nodes corresponding to x and $\overline{x}$. Moreover, an edge connecting the two nodes is added to G.
- For each clause, add three nodes corresponding to the literals. Three edges connecting the three nodes are added to G. Moreover, each of the three node is connected to the node of identical label in the variable gadgets.

VERTEXCOVER is *NP*-Complete

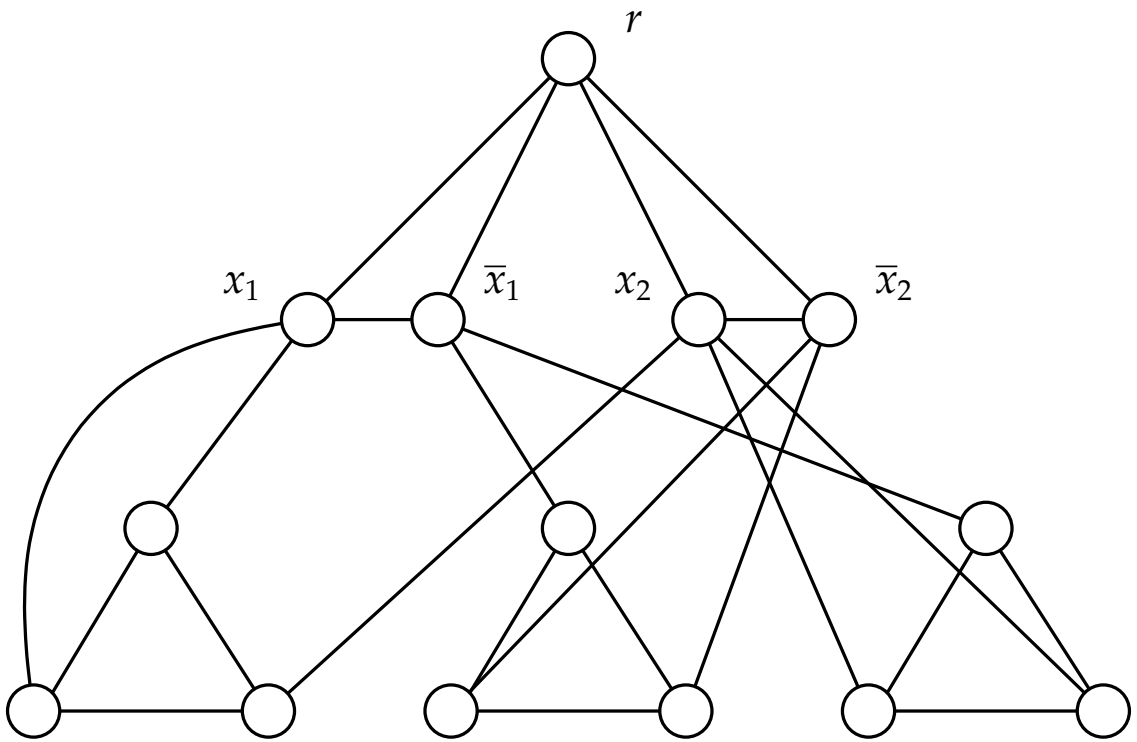
Proof (cont'd).

We show that ϕ is satisfiable if and only if G has an $(l + 2k)$-node vertex cover. Suppose there is a truth assignment satisfying ϕ. Either of the nodes in each variable gadget is 1. We select these l nodes. For each clause gadget, one of the three nodes corresponds to a true literal. We select the other two nodes. These $l + 2k$ nodes form a vertex cover. Conversely, suppose G has a vertex cover of $l + 2k$ nodes. Since there is an edge in every variable gadget, one node in each variable gadget is select. Moreover, two nodes are needed to cover the three edges in each clause gadget. Hence l nodes are from the variable gadgets and $2k$ nodes are from the clause gadgets. We assign the l literals in the variable gadgets to 1. For each clause gadget, only two nodes are in the vertex cover. Observe that each node in a clause gadget has three edges. To cover the three edges connected to the node not in the vertex cover, the literal corresponding to the node must be 1. □

3COLOR

- A coloring of a graph is an assignment of colors to its nodes so that adjacent nodes have different colors. Define

$$3COLOR = \{\langle G \rangle : G \text{ has a coloring with 3 colors}\}.$$



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

3COLOR is NP-Complete

Theorem 33

3COLOR is NP-complete.

Proof.

We reduce $\neq SAT$ to 3COLOR. Let r be a node. A variable gadget for x has two nodes (labeled x and $\overline{x}$) with edges $\{x, \overline{x}\}$, $\{r, x\}$, and $\{r, \overline{x}\}$. A clause gadget for $l_0 \vee l_1 \vee l_2$ has three nodes labeled l_0, l_1, l_2 with edges $\{l_0, l_1\}$, $\{l_0, l_2\}$, $\{l_1, l_2\}$. Moreover, each l_i is connected to the corresponding node in variable gadgets. □

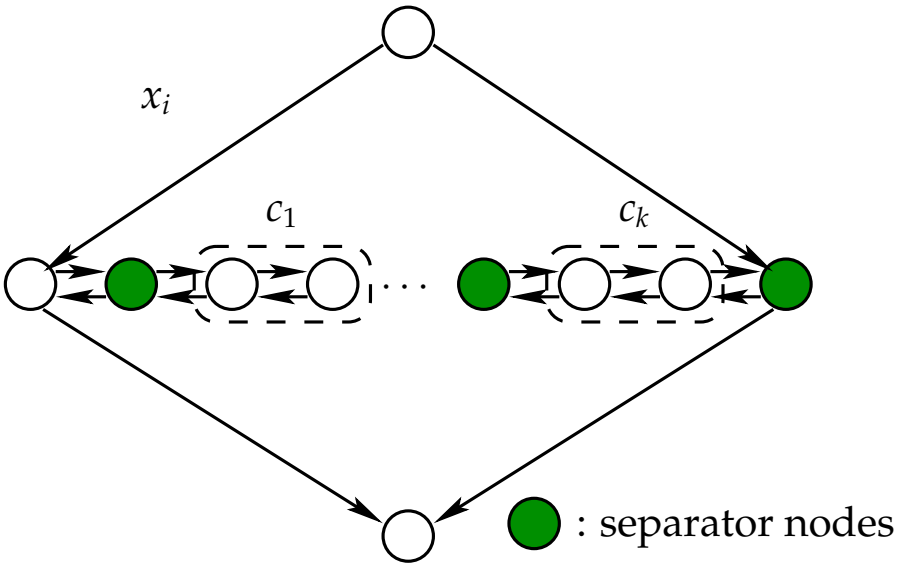
HAMPATH is *NP*-Complete

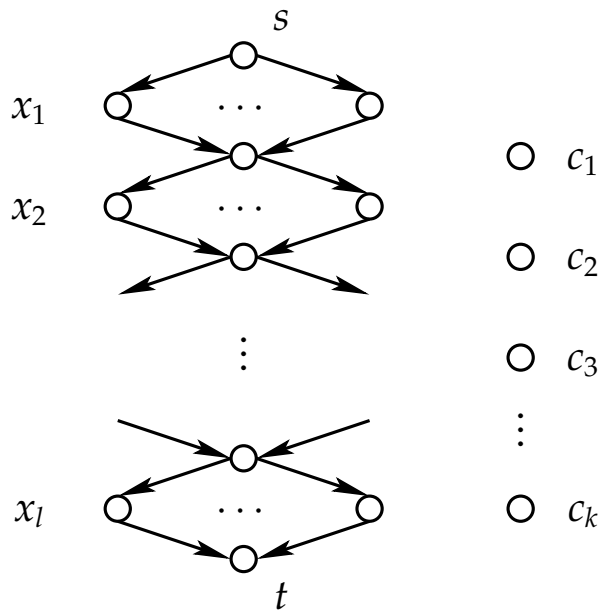
Theorem 34

HAMPATH is NP-complete.

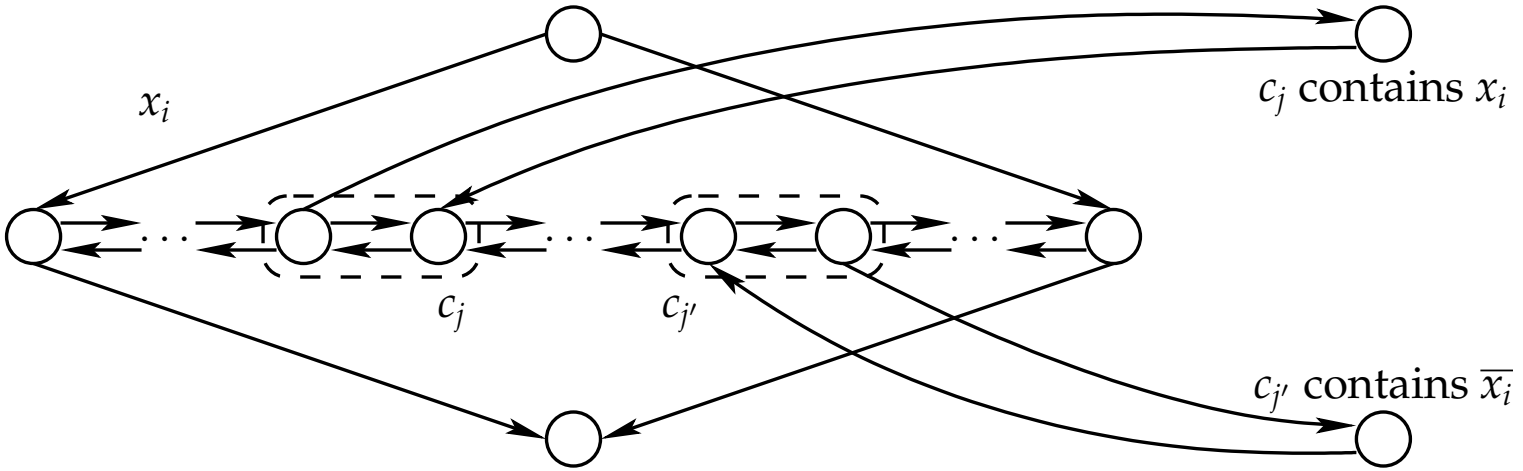
Proof.

We show $3SAT \leq_P HAMPATH$. Let ϕ be a 3CNF-formula with l variables and k clauses. We want to construct a graph G with two nodes s and t such that ϕ is satisfiable if and only G has a Hamiltonian path from s to t.





HAMPATH is *NP*-Complete



Proof (cont'd).

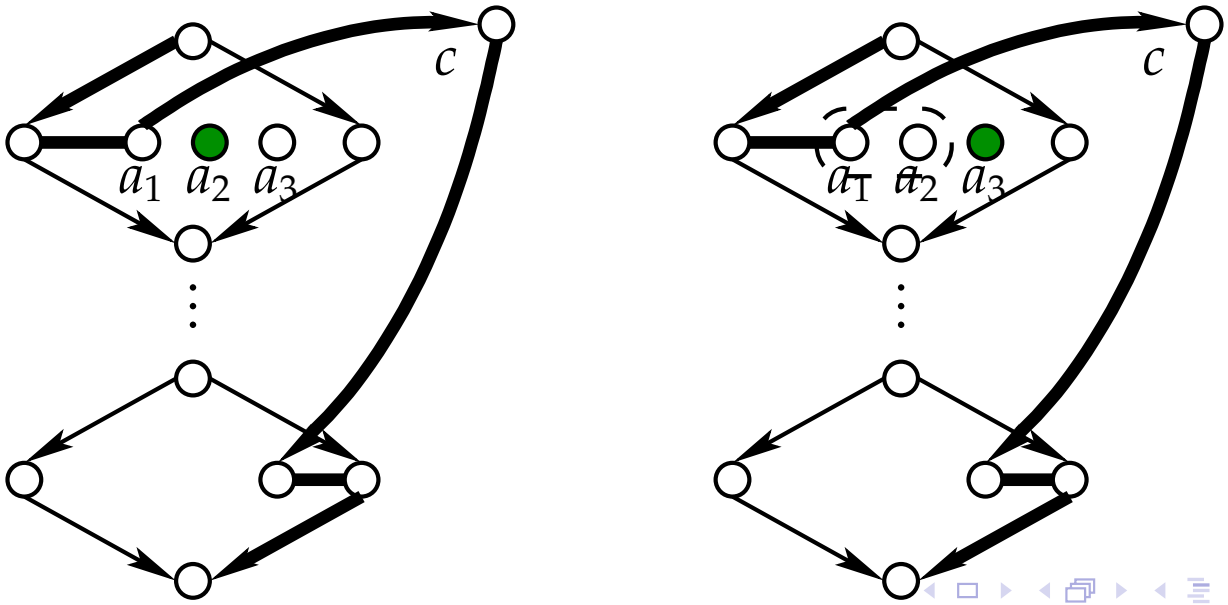
Each variable gadget has $4 + 3k$ nodes. Note that a Hamltonian path traverses a variable gadget in one of the two ways: from left to right, or from right to left.
Each clause gadget has 1 node. If x_i appears in c_j, two edges are added to visit the clause gadget c_j from left to right. If $\overline{x_i}$ appears in $c_{j'}$, two edges are added to visit the clause gadget $c_{j'}$ from right to left.
We show that ϕ is satisfiable if and only if there is a Hamiltonian path from s to t in G.
Suppose ϕ has a satisfying assignment. We start from s and go through each variable gadget in turn. If x_i is assigned to 1, we traverse x_i's gadget from left to right. If x_i is assigned to 0, we traverse x_i's gadget from right to left. For each clause, one of its literal is 1. We detour to its gadget when traversing variable gadgets.

HAMPATH is *NP*-Complete

Proof (cont'd).

Conversely, suppose G has a Hamiltonian path from s to t. We argue that the Hamiltonian path must traverse every variable gadgets and detour to every clause gadget on its way. Suppose not. Then a_2 or a_3 is a separator node.

- If a_2 is a separator node, it is not connected to any clause gadget. The path must visit a_2 from a_3 since a_1 has been visited. But the path would end in a_2.
- If a_3 is a separator node, a_1 and a_2 are in the same clause pair. a_2 must be visited from a_3 since a_1 and c have been visited. But the path would end in a_2 as well. □



UHAMPATH is *NP*-Complete

- Consider

$$UHAMPATH = \{\langle G, s, t\rangle : \quad G \text{ is an undirected graph with a Hamiltonian path from } s \text{ to } t\}.$$

Theorem 35

UHAMPATH is NP-complete.

Proof.

We show $HAMPATH \leq_P UHAMPATH$. Let G be a directed graph. We construct an undirected graph G' as follows.

- For each node u in G, we have three nodes u^{in}, u^{mid}, and u^{out} in G'. Moreover, G' has edges $\{u^{\text{in}}, u^{\text{mid}}\}$ and $\{u^{\text{mid}}, u^{\text{out}}\}$.
- For each edge (u, v) in G, add the edge $\{u^{\text{out}}, v^{\text{in}}\}$ to G'.

We now show that G has a Hamiltonian path from s to t if and only if G' has a Hamiltonian path from s^{in} to t^{out}.

UHAMPATH is *NP*-Complete

Proof.

For a Hamiltonian path $s, u_1, u_2, \ldots, u_k, t$ in G, $s^{\text{in}}, s^{\text{mid}}, s^{\text{out}}, u_1^{\text{in}}, u_1^{\text{mid}}, u_1^{\text{out}}, u_2^{\text{in}}, \ldots, u_k^{\text{in}}, u_k^{\text{mid}}, u_k^{\text{out}}, t^{\text{in}}, t^{\text{mid}}, t^{\text{out}}$ is a Hamiltonian path in G'.
Conversely, any Hamiltonian path in G' from s^{in} must be of the form

$$s^{\text{in}}, s^{\text{mid}}, s^{\text{out}}, u_i^{\text{in}}, \ldots, (\text{for some } i).$$

But this corresponds to the edge (s, u_i) in G. Moreover, the next node must be u_i^{mid}, and then followed by u_i^{out}. The Hamiltonian path now looks like

$$s^{\text{in}}, s^{\text{mid}}, s^{\text{out}}, u_i^{\text{in}}, u_i^{\text{mid}}, u_i^{\text{out}}, u_j^{\text{in}}, \ldots, (\text{for some } j).$$

But this corresponds to the edges (s, u_i) and (u_i, u_j) in G. Eventually we reach t^{out} and obtain a Hamiltonian path from s to t in G. □

SUBSETSUM is *NP*-Complete

Theorem 36

SUBSETSUM is NP-complete.

Proof.

We show $3SAT \leq_P SUBSETSUM$. Let ϕ be a 3CNF-formula with l variables and k clauses. We construct a multiset S so that ϕ is satisfiable if and only if the sum of a subset of S is $t = \overbrace{1\cdots1}^{l}\overbrace{3\cdots3}^{k}$ (a very large decimal number).

- For each variable x_i, we add numbers y_i and z_i to S if x_i appears in c_r, c_t and $\overline{x_i}$ appears in c_s (unspecified digits are 0):

		$\cdots$	i	$\cdots$	$\cdots$	c_r	$\cdots$	c_s	$\cdots$	c_t	$\cdots$
y_i	$=$		1			1				1	
z_i	$=$		1					1			

- For each clause c_j, add g_j and h_j to S (unspecified digits are 0):

			$\cdots$	$\cdots$	c_j	$\cdots$
g_j	$=$	h_j $=$			1	

SUBSETSUM is *NP*-Complete

	1	2	3	4	$\cdots$	l	c_1	c_2	$\cdots$	c_k
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2	0	1	0	0	$\cdots$	0	0	1	$\cdots$	0
z_2	0	1	0	0	$\cdots$	0	1	0	$\cdots$	0
y_3	0	0	1	0	$\cdots$	0	1	1	$\cdots$	0
z_3	0	0	1	0	$\cdots$	0	0	0	$\cdots$	1
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2							0	1	$\cdots$	0
h_2							0	1	$\cdots$	0
$\vdots$										$\vdots$
g_k							0	0	$\cdots$	1
h_k							0	0	$\cdots$	1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

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

SUBSETSUM is *NP*-Complete

Proof.

Suppose ϕ has a satisfying truth assignment. We put y_i to the subset if x_i is assigned to 1; we put z_i if x_i is assigned to 0. Since either y_i or z_i is selected for each i, the first l digits of the sum are all 1's. For each of the last k digits of the sum, it cannot be 0 since one of the literals in the clause is 1. We add g_i's or h_i's to the subset so that the last k digits of the sum are all 3's.

Conversely, suppose S has a subset whose sum is t. Observe that each digit has at most 5 1's. There cannot be any carry. Either y_i or z_i must be selected for each i since the first l digits are all 1's. This gives us a truth assignment for ϕ. Finally, note that the last k digits are all 3's. Yet g_j and h_j contribute at most 2 for each of the last k digits. Subsequently, y_i's or z_i's contribute at least 1 for each of the last k digits. The truth assignment does satisfy ϕ. □