Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

1. NP and NP-Completeness
    - The Class NP
    - Reducibility and NP-Completeness
    - The Cook-Levin Theorem
    - The Web of Reductions
    - Decision Versus Search
    - coNP, EXP and NEXP

Subsection 1

The Class NP

Efficient Verifiability

- We formalize the notion of efficiently verifiable solutions.
- We identified "efficient solvability" with polynomial time.
- So "efficient verifiability" should also correspond to polynomial time.
- By hypothesis, a Turing machine can only read one bit in a step.
- The alleged solution can only be allowed to have at most polynomial length.

The class NP

Definition (The class NP)

A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M (called the **verifier** for L), such that, for every $x \in \{0,1\}^*$,

$$x \in L \;\Leftrightarrow\; \exists u \in \{0,1\}^{p(|x|)}(M(x,u) = 1).$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) = 1$, then we call u a **certificate** for x (with respect to the language L and machine M).

- The term **witness** instead of certificate is often used.
- Clearly, P $\subseteq$ NP, since the polynomial $p(|x|)$ is allowed to be 0, i.e., u can be an empty string.

IndSet is in NP

- We show that the language IndSet is in NP.
- Recall that this language contains all pairs $\langle G, k\rangle$, such that the graph G has a subgraph of at least k vertices with no edges between them.
- Such a subgraph is called an **independent set**.
- Consider the following polynomial-time algorithm M.
  - Input consists of a pair $\langle G, k\rangle$ and a string $u \in \{0,1\}^*$.
  - Output 1 if and only if u encodes a list of k vertices of G, such that there is no edge between any two members of the list.

$\textsc{IndSet}$ is in NP (Cont'd)

- Clearly, $\langle G, k\rangle$ is in $\textsc{IndSet}$ if and only if, there exists a string u, such that
$$M(\langle G, k\rangle, u) = 1.$$
- Hence, $\textsc{IndSet}$ is in NP.
- The list u of k vertices forming the independent set in G serves as the certificate that $\langle G, k\rangle$ is in $\textsc{IndSet}$.
- If n is the number of vertices in G, then a list of k vertices can be encoded using $\mathrm{O}(k \log n)$ bits.
- Thus, u is a string of at most $\mathrm{O}(n \log n)$ bits.
- This is polynomial in the size of the representation of G.

Examples of Decision Problems in NP I

- **Traveling Salesperson**: Given a set of n nodes, $\binom{n}{2}$ numbers $d_{i,j}$ denoting the distances between all pairs of nodes, and a number k, decide if there is a closed circuit (i.e., a "salesperson tour") that visits every node exactly once and has total length at most k.

  The certificate is the sequence of nodes in such a tour.
- **Subset Sum**: Given a list of n numbers $A_1, \ldots, A_n$ and a number T, decide if there is a subset of the numbers that sums up to T.

  The certificate is the list of members in such a subset.
- **Linear Programming**: Given a list of m linear inequalities with rational coefficients over n variables $u_1, \ldots, u_n$ (a linear inequality has the form $a_1 u_1 + a_2 u_2 + \cdots + a_n u_n \leq b$, for some coefficients $a_1, \ldots, a_n, b$), decide if there is an assignment of rational numbers to the variables $u_1, \ldots, u_n$, that satisfies all the inequalities.

  The certificate is the assignment.

Examples of Decision Problems in NP II

- **0/1 Integer Programming**: Given a list of m linear inequalities with rational coefficients over n variables $u_1, \ldots, u_n$, find out if there is an assignment of zeroes and ones to $u_1, \ldots, u_n$ satisfying all the inequalities.

  The certificate is the assignment.

- **Graph Isomorphism**: Given two $n \times n$ adjacency matrices M_1, M_2, decide if M_1 and M_2 define the same graph, up to renaming of vertices.

  The certificate is the permutation $\pi : [n] \to [n]$, such that M_2 is equal to M_1 after reordering M_1's indices according to π.

- **Composite Numbers**: Given a number N decide if N is a composite (i.e., non-prime) number.

  The certificate is the factorization of N.

- **Factoring**: Given three numbers N, L, U, decide if N has a prime factor p in the interval $[L, U]$.

  The certificate is the factor p.

Examples of Decision Problems in NP III

- **Connectivity**: Given a graph G and two vertices s, t in G, decide if s is connected to t in G.

  The certificate is a path from s to t.
- The Status of the Problems:
  - In the preceding list, the Connectivity, Composite Numbers, and Linear Programming problems are known to be in P.
  - All the other problems in the list are not known to be in P, but no proof of nonmembership exists.
  - The Independent Set, Traveling Salesperson, Subset Sum, and Integer Programming problems are known to be NP-complete, which, implies that they are not in P unless P = NP.
  - The Graph Isomorphism and Factoring problems are not known to be either in P or be NP-complete.

Relation between NP and P

Claim

Let $\mathrm{EXP} = \bigcup_{c \geq 1} 2^{\mathrm{O}(n^c)}$. Then

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}.$$

- We first show that $\mathrm{P} \subseteq \mathrm{NP}$.

  Suppose $L \in \mathrm{P}$ is decided in polynomial time by a TM N.

  Take N as the verifier M, with the polynomial $p(x)$ being the zero polynomial, i.e., u the empty string.

  This shows that $L \in \mathrm{NP}$.

Relation between NP and P (Cont'd)

- We next show that $\mathrm{NP} \subseteq \mathrm{EXP}$.

  Let $L \in \mathrm{NP}$.

  Let M be the verifier for L, with p the associated polynomial.

  Then we can decide L in time $2^{\mathrm{O}(p(n))}$ by:
  - Enumerating all possible strings u;
  - Using M to check whether u is a valid certificate for the input x.

  The machine accepts iff such a u is ever found.

  We have that $p(n) = \mathrm{O}(n^c)$, for some $c \geq 1$.

  So the number of choices for u is $2^{\mathrm{O}(n^c)}$.

  The running time of the machine is similar.

$P \overset{?}{=} NP$

- The question whether or not $P = NP$ is considered the central open question of complexity theory, mathematics and science.
- Most researchers believe that $P \neq NP$.
- The main reason is that years of effort have failed to yield efficient algorithms for NP-complete problems.

Nondeterministic Turing Machines

- The class NP can also be defined using a variant of Turing machines called **nondeterministic Turing machines** (**NDTM**).
- This was the original definition, and the reason for the name NP, which stands for nondeterministic polynomial time.
- The only difference between an NDTM and a standard TM is that an NDTM has two transition functions δ_0 and δ_1, and a special state denoted by q_{accept}.
    - When an NDTM M computes a function, at each computational step M makes an arbitrary choice as to which of its two transition functions to apply.
    - For every input x, we say that $M(x) = 1$ if there exists some sequence of these choices, called the **nondeterministic choices** of M, that would make M reach q_{accept} on input x.
      Otherwise, if every sequence of choices makes M halt without reaching q_{accept}, then we say that $M(x) = 0$.

Nondeterministic Time Classes

- We say that the NDTM M **runs in $T(n)$ time** if, for every input $x \in \{0,1\}^*$ and every sequence of nondeterministic choices, M reaches either the halting state or q_{accept} within $T(|x|)$ steps.

Definition (Nondeterministic Time)

For every function $T : \mathbb{N} \to \mathbb{N}$ and $L \in \{0,1\}^*$, we say that

$$L \in \mathsf{NTIME}(T(n))$$

if, there exists a constant $c > 0$ and a $c \cdot T(n)$-time NDTM M, such that, for every $x \in \{0,1\}^*$,

$$x \in L \iff M(x) = 1.$$

NP Equals Nondeterministic Polynomial Time

Theorem

We have

$$\mathsf{NP} = \bigcup_{c \geq 1} \mathsf{NTIME}(n^c).$$

- The main idea is that the sequence of nondeterministic choices made by an accepting computation of an NDTM can be viewed as a certificate that the input is in the language, and vice versa.

Proof of $\supseteq$

- Suppose $p : \mathbb{N} \rightarrow \mathbb{N}$ is a polynomial and L is decided by a NDTM N that runs in time $p(n)$.

  For every $x \in L$, there is a sequence of nondeterministic choices that makes N reach q_{accept} on input x.

  We can use this sequence as a certificate for x.

  - This certificate has length $p(|x|)$.
  - We use a deterministic machine that simulates in polynomial time the action of N using these nondeterministic choices.
  - The machine verifies that N would have entered q_{accept} after using these nondeterministic choices.

  Thus, $L \in \text{NP}$.

Proof of $\subseteq$

- Conversely, suppose that $L \in \mathrm{NP}$.

  Consider a polynomial time NDTM N that decides L:
    - On input x, use the ability to make nondeterministic choices to write down a string u of length $p(|x|)$.
    - This can be done by having transition δ_0 correspond to writing a 0 on the tape and transition δ_1 correspond to writing a 1.
    - Run the deterministic verifier M to verify that u is a certificate for x.
    - If so, enter q_{accept}.

  Clearly, N enters q_{accept} on x if and only if a certificate exists for x.

  We know that $p(n) = \mathrm{O}(n^c)$, for some $c \geq 1$.

  We conclude that $L \in \mathrm{NTIME}(n^c)$.
- NDTMs can be easily represented as strings, whence there exists a universal nondeterministic Turing machine.

Subsection 2

Reducibility and NP-Completeness

Introducing Reductions

- IndSet is at least as hard as any other language in NP.
- This means that if IndSet has a polynomial time algorithm then so do all the problems in NP.
- The property is called NP-hardness.
- Our conjecture states that NP $\neq$ P.
- So a language being NP-hard provides evidence that it cannot be decided in polynomial time.
- To prove that a language C is at least as hard as some other language B, we introduce the notion of a reduction.

Reductions, NP-hardness and NP-completeness

Definition (Reductions, NP-hardness and NP-completeness)

A language $L \in \{0,1\}^*$ is **polynomial-time Karp reducible**, or simply **polynomial-time reducible**, to a language $L' \in \{0,1\}^*$, denoted by

$$L \leq_p L',$$

if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$, such that, for every $x \in \{0,1\}^*$,

$$x \in L \quad \text{if and only if} \quad f(x) \in L'.$$

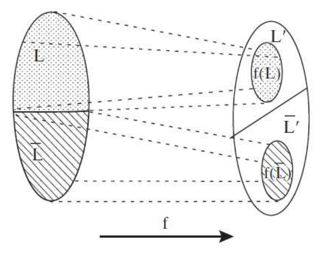
We say that L' is NP-**hard** if,

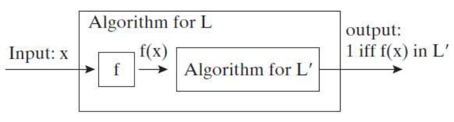
$$L \leq_p L', \quad \text{for every } L \in \text{NP}.$$

We say that L' is NP-**complete** if L' is NP-hard and $L' \in \text{NP}$.

Illustration of Polynomial-Time Karp Reductions

- A Karp reduction from L to L' is a polynomial-time function f that maps strings in L to strings in L' and strings in $\overline{L} = \{0,1\}^* \setminus L$ to strings in $\overline{L'}$.





- f can be used to transform a polynomial time TM M' that decides L' to a polynomial time TM M for L by setting $M(x) = M'(f(x))$.

Properties of Polynomial-Time Karp Reductions

Theorem

1. (**Transitivity**) If $L \leq_p L'$ and $L' \leq_p L''$, then $L \leq_p L''$.
2. If language L is NP-hard and $L \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$.
3. If language L is NP-complete, then $L \in \mathrm{P}$ if and only if $\mathrm{P} = \mathrm{NP}$.

- The main observation underlying all three parts is that if p, q are two functions that grow at most as n^c and n^d, respectively, then their composition $p(q(n))$ grows as at most n^{cd}, which is also polynomial.

1. Let f_1 be a polynomial-time reduction from L to L'.

   Let f_2 be a polynomial-time reduction from L' to L''.

   Then $x \mapsto f_2(f_1(x))$ is a polynomial-time reduction from L to L'':
   - $f_2(f_1(x))$ takes polynomial time to compute given x.
   - We have
   $$f_2(f_1(x)) \in L'' \quad \text{iff} \quad f_1(x) \in L' \quad \text{iff} \quad x \in L.$$

Properties of Polynomial-Time Karp Reductions (Cont'd)

2. We know $\mathrm{P} \subseteq \mathrm{NP}$.

   Assume $L' \in \mathrm{NP}$.

   Since L is NP-hard, $L' \leq_p L$.

   Since $L \in \mathrm{P}$, then $L' \in \mathrm{P}$.

   So $\mathrm{NP} \subseteq \mathrm{P}$.
3. For left-to-right, suppose L is NP-complete.

   Then it is NP-hard.

   Thus, by Property 2, $\mathrm{P} = \mathrm{NP}$.

   For right-to-left, assume $\mathrm{NP} = \mathrm{P}$.

   Since L is NP-complete, $L \in \mathrm{NP}$.

   Thus, by hypothesis, $L \in \mathrm{P}$.

The First NP-Complete Language

- We show that NP-complete languages exist by exhibiting a language in NP that is as hard as any other language in NP.

Theorem

The following language is NP-complete:

$$\begin{aligned}\textsc{TMSat} \quad = \quad \{\langle \alpha, x, 1^n, 1^t\rangle : \exists u \in \{0,1\}^n (&M_\alpha \text{ outputs } 1 \text{ on input} \\ &\langle x, u\rangle \text{ within } t \text{ steps})\},\end{aligned}$$

where M_α denotes the (deterministic) TM represented by the string α.

- Let L be an NP-language.

  Then, there is a polynomial p and a verifier TM M such that $x \in L$ iff:
  - There is a string $u \in \{0,1\}^{p(|x|)}$ satisfying $M(x,u) = 1$;
  - M runs in time $q(n)$, for some polynomial q.

The First NP-Complete Language (Cont'd)

- We reduce L to $\textsc{TMSat}$.

  We map every string $x \in \{0,1\}^*$ to the tuple

  $$\langle \llcorner M \lrcorner, x, 1^{p(|x|)}, 1^{q(m)} \rangle,$$

  where:

  - $m = |x| + p(|x|)$;
  - $\llcorner M \lrcorner$ denotes the representation of M as a string.

  This mapping can clearly be performed in polynomial time.

  Moreover, by the definition of $\textsc{TMSat}$ and the choice of M,

  $$\begin{array}{ll} \multicolumn{2}{l}{\langle \llcorner M \lrcorner, x, 1^{p(|x|)}, 1^{q(m)} \rangle \in \textsc{TMSat}} \\ \text{iff} & \exists u \in \{0,1\}^{p(|x|)} (M(x,u) \text{ outputs 1 within } q(m) \text{ steps}) \\ \text{iff} & x \in L. \end{array}$$

Subsection 3

The Cook-Levin Theorem

Satisfiable Boolean Formulas

- A **Boolean formula** over the variables $u_1, \ldots, u_n$ consists of the variables and the logical operators AND ($\wedge$), OR ($\vee$) and NOT ($\neg$).

  Example: $(u_1 \wedge u_2) \vee (u_2 \wedge u_3) \vee (u_3 \wedge u_1)$ is a Boolean formula.
- If φ is a Boolean formula over variables $u_1, \ldots, u_n$ and $z \in \{0, 1\}^n$, then $\varphi(z)$ denotes the **value** of φ when the variables of φ are assigned the values z (where we identify 1 with TRUE and 0 with FALSE).
- A formula φ is **satisfiable** if there exists some assignment z such that $\varphi(z)$ is TRUE.
- Otherwise, we say that φ is **unsatisfiable**.

  Example: $(u_1 \wedge u_2) \vee (u_2 \wedge u_3) \vee (u_3 \wedge u_1)$ is satisfiable, since the assignment $u_1 = 1$, $u_2 = 0$, $u_3 = 1$ satisfies it.

  In general, an assignment $u_1 = z_1$, $u_2 = z_2$, $u_3 = z_3$ satisfies this formula iff at least two of the z_i's are 1.

Conjunctive Normal Form, SAT and 3SAT

- A Boolean formula over variables $u_1, \ldots, u_n$ is in **CNF form** (shorthand for **Conjunctive Normal Form**) if it is an AND of OR's of variables or their negations.

  Example: The following is a 3CNF formula, where $\overline{u}_i$ denotes $\neg u_i$,

  $$(u_1 \vee \overline{u}_2 \vee u_3) \wedge (u_2 \vee \overline{u}_3 \vee u_4) \wedge (\overline{u}_1 \vee u_3 \vee \overline{u}_4).$$

- More generally, a **CNF formula** has the form

  $$\bigwedge_i \left(\bigvee_j v_{i_j} \right),$$

  where each v_{i_j} is either a variable u_k or its negation $\overline{u}_k$.

Conjunctive Normal Form, $\textsc{Sat}$ and $3\textsc{Sat}$ (Cont'd)

- Consider a CNF formula

$$\bigwedge_i \left(\bigvee_j v_{i_j} \right).$$

  - The terms v_{i_j} are called its **literals**;
  - The terms $(\bigvee_j v_{i_j})$ are called its **clauses**.
- The **size** of a CNF formula is defined to be the number of $\wedge/\vee$ symbols it contains.
- A **kCNF** is a CNF formula with all clauses having at most k literals.
- We denote by $\textsc{Sat}$ the language of all satisfiable CNF formulae.
- We denote by $3\textsc{Sat}$ the language of all satisfiable 3CNF formulae.

Expressing Equality of Strings

- The formula $(x_1 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee y_1)$ is in CNF form.
- It is satisfied by only those values of x_1, y_1 that are equal.
- Thus, the formula

$$(x_1 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee y_1) \wedge \cdots \wedge (x_n \vee \overline{y}_n) \wedge (\overline{x}_n \vee y_n)$$

  is satisfied by an assignment if and only if each x_i is assigned the same value as y_i.
- Thus, though $=$ is not a standard Boolean operator like $\vee$ or $\wedge$, we will use it as a convenient shorthand, since the formula $\phi_1 = \phi_2$ is equivalent to (has the same satisfying assignments as)

$$(\phi_1 \vee \overline{\phi}_2) \wedge (\overline{\phi}_1 \vee \phi_2).$$

Universality of AND, OR, NOT

Claim

For every Boolean function $f : \{0,1\}^\ell \to \{0,1\}$, there is an ℓ-variable CNF formula φ of size $\ell 2^\ell$, such that

$$\varphi(u) = f(u), \quad \text{for every } u \in \{0,1\}^\ell.$$

- For every $v \in \{0,1\}^\ell$, there exists a clause $C_v(z_1, z_2, \ldots, z_\ell)$ in ℓ variables, such that $C_v(v) = 0$ and $C_v(u) = 1$, for every $u \neq v$.

  E.g., if $v = \langle 1,1,0,1 \rangle$, the corresponding clause is

  $$\overline{z}_1 \vee \overline{z}_2 \vee z_3 \vee \overline{z}_4.$$

  Let φ be the AND of all the clauses C_v, for v such that $f(v) = 0$.

Universality of AND, OR, NOT (Cont'd)

- We set

  $$\varphi = \bigwedge_{v : f(v) = 0} C_v(z_1, z_2, \ldots, z_\ell).$$

  φ has size at most $\ell 2^\ell$.

  Consider an assignment $u \in \{0, 1\}^\ell$.
  - Suppose, fist, that $f(u) = 0$.
    Then $C_u(u) = 0$.
    So $\varphi(u) = 0$.
  - Suppose, on the other hand, that $f(u) = 1$.
    Then $C_v(u) = 1$, for every v.
    Therefore, $\varphi(u) = 1$.

  Thus, for every $u \in \{0, 1\}^\ell$, $\varphi(u) = f(u)$.

The Cook-Levin Theorem

- The following theorem provides the first natural NP-complete problem.

Theorem (Cook-Levin Theorem)

1. SAT is NP-complete.
2. 3SAT is NP-complete.

- Both SAT and 3SAT are clearly in NP, since a satisfying assignment can serve as the certificate that a formula is satisfiable.

  Thus we only need to prove that they are NP-hard.

  We do so by:

  (a) Proving that SAT is NP-hard;
  (b) Showing that SAT is polynomial-time Karp reducible to 3SAT.
  By the transitivity of polynomial-time reductions, this implies that 3SAT is NP-hard.

NP-hardness of Satisfiability

Lemma

$\textsc{Sat}$ is NP-hard.

- We must show how to reduce every NP language L to $\textsc{Sat}$.

  We need a polynomial-time transformation that turns any $x \in \{0,1\}^*$ into a CNF formula φ_x, such that

  $$x \in L \quad \text{iff} \quad \varphi_x \text{ is satisfiable.}$$

  $L \in \mathrm{NP}$ means that, there exists a polynomial time TM M, such that, for every $x \in \{0,1\}^*$,

  $$x \in L \quad \text{iff} \quad M(x,u) = 1, \text{ for some } u \in \{0,1\}^{p(|x|)},$$

  where $p : \mathbb{N} \to \mathbb{N}$ is some polynomial.

NP-hardness of Satisfiability (Cont'd)

- We describe a polynomial-time transformation $x \mapsto \varphi_x$ from strings to CNF formulae, such that

$$x \in L \quad \text{iff} \quad \varphi_x \text{ is satisfiable.}$$

Equivalently,

$$\varphi_x \in \textsc{Sat} \quad \text{iff} \quad \exists u \in \{0,1\}^{p(|x|)}(M(x \circ u) = 1).$$

To get a polynomial size formula, we use he following facts:

  - M runs in polynomial time;
  - Each basic step of a Turing machine is highly local.
    That is, it examines and changes only a few bits of the machine's tapes.

Simplifying Assumptions About M

- In the course of the proof, we will make the following simplifying assumptions about the TM M:
  (i) M only has two tapes, an input tape and a work/output tape.
  (ii) M is an oblivious TM in the sense that its head movement does not depend on the contents of its tapes.
  This means that:
    - M's computation takes the same time for all inputs of size n;
    - For every i, the location of M's heads at the ith step depends only on i and the length of the input.

  We can make these assumptions without loss of generality because, for every $T(n)$-time TM M, there exists a two-tape oblivious TM $\widetilde{M}$ computing the same function in $O\left(T(n)^2\right)$ time.

Simplifying Assumptions About M (Cont'd)

- In particular, if L is in NP, then there exists a two-tape oblivious polynomial-time TM M and a polynomial p, such that

  $$x \in L \quad \text{iff} \quad \exists u \in \{0,1\}^{p(|x|)}(M(x \circ u) = 1).$$

  Since M is oblivious, we can run it on the trivial input $(x, 0^{p(|x|)})$ to determine the precise head position of M during its computation on every other input of the same length.

Snapshot of M's Computation

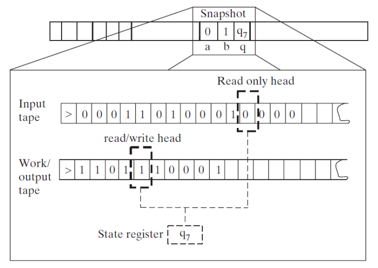
- Let Q be the set of M's possible states.

  Let Γ be M's alphabet.

  The **snapshot** of M's execution on input y at a particular step i is the triple

  $$\langle a, b, q\rangle \in \Gamma \times \Gamma \times Q,$$

  such that:
  - a, b are the symbols read by M's heads from the two tapes;
  - q is the state M is in at the ith step.



Encoding the Snapshot

- Clearly the snapshot can be encoded as a binary string.

  Let c denote the length of this string, a constant depending upon $|Q|$ and $|\Gamma|$.

  For every $y \in \{0,1\}^*$, the snapshot of M's execution on input y at the ith step depends on:

  (a) Its state in the $(i-1)$st step;
  (b) The contents of the current cells of its input and work tapes.

Insight at the Heart of the Proof

- We want to verify the existence of some u satisfying $M(x \circ u) = 1$.

  We are given, as evidence, the sequence of snapshots that arise from M's execution on $x \circ u$.

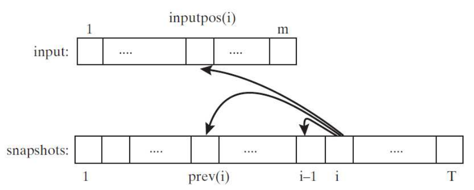
  It suffices to check that, for each $i \leq T(n)$, the snapshot z_i is correct given the snapshots for the previous $i - 1$ steps.

  The TM can only read/modify one bit at a time.

  It follows that, to check the correctness of z_i, it suffices to look at only two of the previous snapshots.

Insight at the Heart of the Proof (Cont'd)

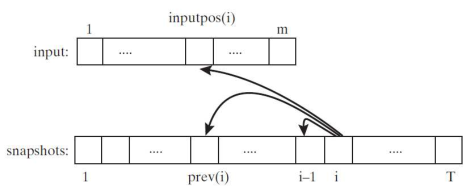
- Specifically, to check z_i we need to only look at z_{i-1}, $y_{\text{inputpos}(i)}$, $z_{\text{prev}(i)}$, where:



  - y is shorthand for $x \circ u$;
  - inputpos(i) denotes the location of M's input tape head at the ith step;
  - prev(i) is the last step before i when M's head was in the same cell on its work tape that it is on during step i.

  Note the contents of the current cell have not been affected between step prev(i) and step i.

Insight at the Heart of the Proof (Cont'd)



- Since M is a deterministic TM, for every triple of values to z_{i-1}, $y_{\text{inputpos}(i)}$, $z_{\text{prev}(i)}$, there is at most one value of z_i that is correct.

  Thus, there is some function F, derived from M's transition function, that maps $\{0,1\}^{2c+1}$ to $\{0,1\}^c$ such that a correct z_i satisfies

  $$z_i = F(z_{i-1}, z_{\text{prev}(i)}, y_{\text{inputpos}(i)}).$$

  Because M is oblivious, the values inputpos(i) and prev(i) do not depend on the particular input y.

  Moreover, these indices can be computed in polynomial-time by simulating M on a trivial input.

The Reduction

- Input $x \in \{0,1\}^n$ is in L iff $M(x \circ u) = 1$, for some $u \in \{0,1\}^{p(n)}$.

  This occurs if and only if there exists a string $y \in \{0,1\}^{n+p(n)}$ and a sequence of strings $z_1, \ldots, z_{T(n)} \in \{0,1\}^c$, where $T(n)$ is the number of steps M takes on inputs of length $n + p(n)$, satisfying the following four conditions.

  1. The first n bits of y are equal to x.
  2. The string z_1 encodes the initial snapshot of M, i.e., z_1 encodes the triple $\langle \triangleright, \sqcup, q_{\text{start}} \rangle$, where $\triangleright$ is the start symbol of the input tape, $\sqcup$ is the blank symbol, and q_{start} is the initial state of the TM M.
  3. For every $i \in \{2, \ldots, T(n)\}$, $z_i = F(z_{i-1}, z_{\text{inputpos}(i)}, z_{\text{prev}(i)})$.
  4. The last string $z_{T(n)}$ encodes a snapshot in which the machine halts and outputs 1.

  The formula φ_x will take $y \in \{0,1\}^{n+p(n)}$ and $z \in \{0,1\}^{cT(n)}$ and will verify that y, z satisfy the AND of these four conditions.

  Thus, $x \in L$ iff $\varphi_x \in \textsc{Sat}$.

Polynomiality of the Reduction

- We can express φ_x as a polynomial-sized CNF formula:
  - Condition 1 can be expressed as a CNF formula of size $4n$;
  - Conditions 2 and 4 each depend on c variables and, hence, can be expressed by CNF formulae of size $c2^c$;
  - Condition 3, which is an AND of $T(n)$ conditions each depending on at most $3c+1$ variables, can be expressed as a CNF formula of size at most $T(n)(3c+1)2^{3c+1}$.

  Hence, the AND of all these conditions can be expressed as a CNF formula of size $d(n+T(n))$, where d is some constant depending only on M.

  Moreover, this CNF formula can be computed in time polynomial in the running time of M.

Reducing $\textsc{Sat}$ to $3\textsc{Sat}$

Lemma

$\textsc{Sat} \leq_p 3\textsc{Sat}$.

- We give a transformation that maps each CNF formula φ into a 3CNF formula ψ, such that ψ is satisfiable if and only if φ is.

  We consider the case where φ is a 4CNF.

  Let C be a clause of φ, say

  $$C = u_1 \vee \overline{u}_2 \vee \overline{u}_3 \vee u_4.$$

  We add a new variable z to φ.

  We replace C with the pair of clauses

  $$\begin{array}{rcl} C_1 & = & u_1 \vee \overline{u}_2 \vee z; \\ C_2 & = & \overline{u}_3 \vee u_4 \vee \overline{z}. \end{array}$$

Reducing $\textsc{Sat}$ to $3\textsc{Sat}$ (Cont'd)

- We verify the equivalence by considering two cases.
  - Suppose $u_1 \vee \overline{u}_2 \vee \overline{u}_3 \vee u_4$ is true.
    Then there is an assignment to z that satisfies both $u_1 \vee \overline{u}_2 \vee z$ and $\overline{u}_3 \vee u_4 \vee \overline{z}$ and vice versa.
  - Suppose C is false.
    Then, no matter what value we assign to z, either C_1 or C_2 will be false.

The same idea can be applied to a general clause of size 4.

It can be used to change every clause C of size k (for $k > 3$) into an equivalent pair of clauses C_1, of size $k - 1$, and C_2, of size 3.

C_1 and C_2 depend on the k variables of C and an additional auxiliary variable z.

Applying this transformation repeatedly, we get a polynomial-time transformation of a CNF formula φ into an equivalent 3CNF formula ψ.

Refinement: Size of φ_x

- The proof of the Cook-Levin Theorem actually yields a result that is a bit stronger than the theorem's statement:

1. We can reduce the size of the output formula φ_x if we use a more efficient simulation of a standard TM by an oblivious TM, which manages to keep the simulation overhead logarithmic.

   Then, for every $x \in \{0,1\}^*$, the size of the formula φ_x is $O(T \log T)$, where T is the number of steps the machine M takes on input x.

Refinement: Levin and Parsimonious Reductions

2. The reduction f from an NP-language L to $\textsc{Sat}$, not only satisfies that

$$x \in L \quad \text{iff} \quad f(x) \in \textsc{Sat},$$

but actually the proof yields an efficient way to transform a certificate for x to a satisfying assignment for $f(x)$ and vice versa.

We call a reduction with this property a **Levin reduction**.

One can also modify the proof slightly so that it actually supplies us with a one-to-one and onto map between the set of certificates for x and the set of satisfying assignments for $f(x)$

Such a construction implies, of course, that the set of certificates for x and the set of satisfying assignments for $f(x)$ are of the same size.

A reduction with this property is called **parsimonious**.

Most of the known NP-complete problems have parsimonious Levin reductions from all the NP-languages.

Importance of the NP-Completeness of 3SAT

- The fact that 3SAT is NP-complete is of paramount interest for several reasons.
  - 3SAT is useful for proving the NP-completeness of other problems.
    It has very minimal combinatorial structure and, thus, is easy to use in reductions.
  - Propositional logic has had a central role in mathematical logic.
    This is why Cook and Levin were interested in 3SAT in the first place.
  - 3SAT has practical importance.
    It is a simple example of constraint satisfaction problems, which are ubiquitous in many fields including artificial intelligence.

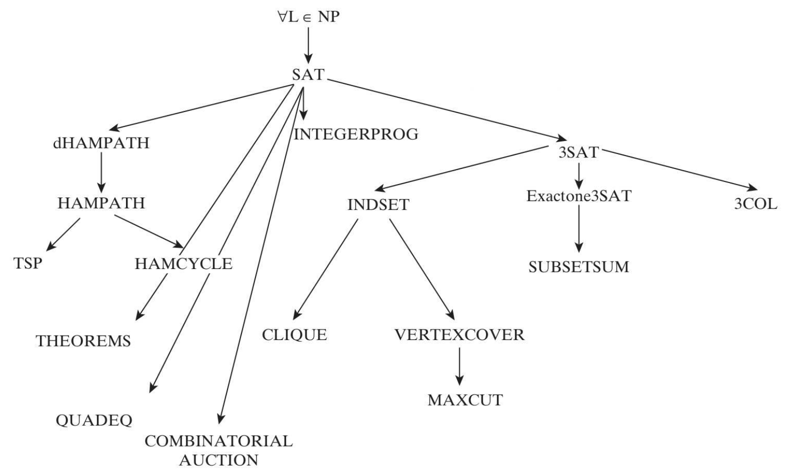
Subsection 4

The Web of Reductions

Web of Reductions

- We showed that $\textsc{Sat}$ and $3\textsc{Sat}$ are NP-complete.
- So to prove the NP-completeness of any other language L, we need to reduce $\textsc{Sat}$ or $3\textsc{Sat}$ to L.
- In fact, once we know that L is NP-complete, we can show that an NP-language L' is NP-complete by reducing L to L'.
- This approach is used to build a “web of reductions”.
- The method is used to show that thousands of interesting languages are in fact NP-complete.

Part of the Web of Reductions



NP-Completeness of IndSet

- Consider

$$\textsc{IndSet} = \{\langle G, k\rangle : G \text{ has independent set of size } k\}.$$

Theorem

IndSet is NP-complete.

- We know IndSet is in NP.

  To show that it is NP-hard, we reduce 3Sat to IndSet.

  We transform, in polynomial time, every m-clause 3CNF formula φ into a $7m$-vertex graph G, such that

$$\varphi \text{ is satisfiable} \quad \text{if and only if} \quad G \text{ has an independent set of size at least } m.$$

Construction of the Graph G

- The graph G is defined as follows.
- We associate a cluster of 7 vertices in G with each clause of φ.

  The vertices in a cluster associated with a clause C correspond to the seven possible satisfying partial assignments to the three variables on which C depends.

  For example, suppose C is

  $$\overline{u}_2 \vee \overline{u}_5 \vee u_7.$$

  Then the seven vertices in the cluster associated with C correspond to all partial assignments of the form

  $$u_1 = a,\ u_2 = b,\ u_3 = c, \quad \langle a, b, c\rangle \neq \langle 1, 1, 0\rangle.$$

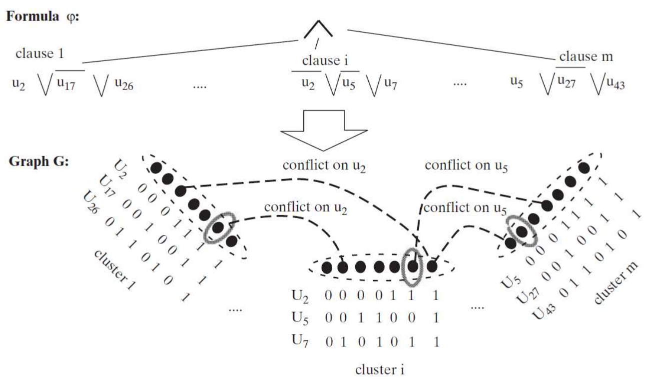
  In case C depends on less than three variables, then we repeat one of the partial assignments.

Completing the Construction

- We put an edge between two vertices of G if they correspond to inconsistent partial assignments.

  E.g., $u_2 = 0$, $u_{17} = 1$, $u_{26} = 1$ and $u_2 = 1$, $u_5 = 0$, $u_7 = 1$.

  We also put edges between every two vertices in the same cluster.



Polynomiality and Correctness

- Transforming φ into G can be done in polynomial time.

  We show that φ is satisfiable iff G has an independent set of size m.

  Suppose, first, that φ has a satisfying assignment u.

  We define a set S of m of G's vertices.

  For every clause C of φ, put in S the vertex in the cluster associated with C that corresponds to the restriction of u to the variables C depends on.

  We only choose vertices that correspond to restrictions of the assignment u.

  So no two vertices of S correspond to inconsistent assignments.

  Hence, S is an independent set of size m.

Correctness (Cont'd)

- Suppose, conversely, that G has an independent set S of size m.

  We define a satisfying assignment u for φ.

  For every $i \in [n]$:

  - If there is a vertex in S whose partial assignment gives a value a to u_i, then set $u_i = a$;
  - Otherwise, set $u_i = 0$.

  This is well defined because of S's independence.

  By construction, G contains all the edges within each cluster.

  So S can contain at most a single vertex in each cluster.

  Thus, there is an element of S in every one of the m clusters.

  Thus, by our definition of u, u satisfies all of φ's clauses.

0/1 Integer Programming

- Let $0/1\textsc{IProg}$ be the set of satisfiable 0/1 Integer Programs,
- I.e., a set of linear inequalities with rational coefficients over variables $u_1, \ldots, u_n$ is in $0/1\textsc{IProg}$ if there is an assignment of numbers in $\{0, 1\}$ to $u_1, \ldots, u_n$ that satisfies it.

Theorem

$0/1\textsc{IProg}$ is NP-complete.

- $0/1\textsc{IProg}$ is in NP, since the assignment can serve as the certificate.
- To reduce $\textsc{Sat}$ to $0/1\textsc{IProg}$, note that every CNF formula can be easily expressed as an integer program.

  This is done by expressing every clause as an inequality.

  E.g., $u_1 \vee \overline{u}_2 \vee \overline{u}_3$ can be expressed as $u_1 + (1 - u_2) + (1 - u_3) \geq 1$.

NP-Completeness of Hamiltonian Path

- A **Hamiltonian path** in a directed graph is a path that visits all vertices exactly once.
- Let dHamPath denote the set of all directed graphs that contain such a path.

Theorem

dHamPath is NP-complete.

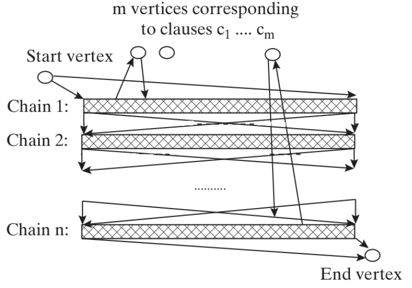
- dHamPath is in NP, since the ordered list of vertices in the path can serve as a certificate.

  To show that dHamPath is NP-hard, we show a way to map every CNF formula φ into a graph G, such that

  $$\varphi \text{ is satisfiable} \quad \text{if and only if} \quad G \text{ has a Hamiltonian path.}$$

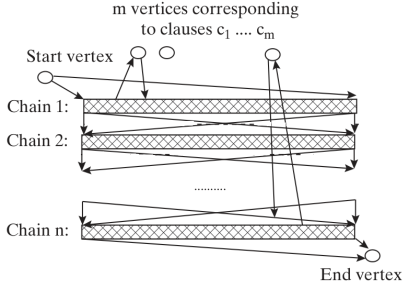
NP-Completeness of Hamiltonian Path (Cont'd)

- The graph G has:
  1. m vertices for each of φ's clauses $c_1, \ldots, c_m$;
  2. A special starting vertex v_{start} and ending vertex v_{end};
  3. n "chains" of $4m$ vertices corresponding to the n variables of φ.
     A **chain** is a set of vertices $v_1, \ldots, v_{4m}$, such that, for every $i \in [4m-1]$, v_i and v_{i+1} are connected by two edges in both directions.



NP-Completeness of Hamiltonian Path (Cont'd)

- We put edges from the starting vertex v_{start} to the two extreme points of the first chain.



We also put edges from the extreme points of the jth chain to the extreme points to the $(j+1)$st chain, for every $j \in [n-1]$.

We put an edge from the extreme points of the nth chain to the ending vertex v_{end}.

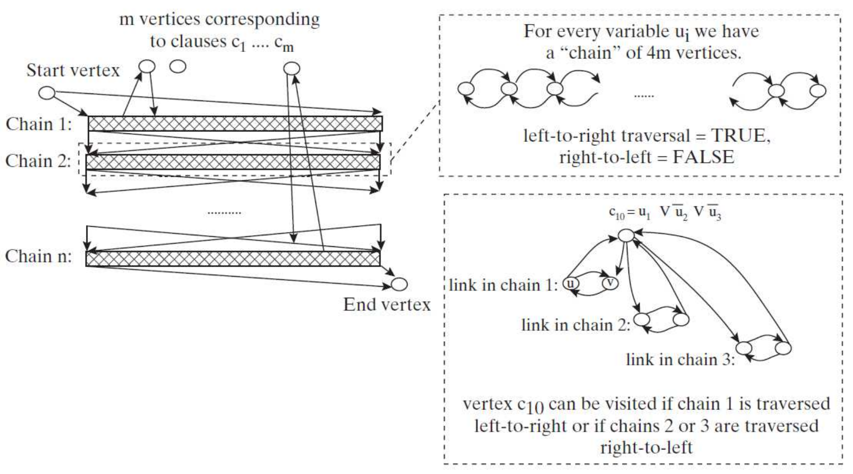
Completing the Construction

- For every clause C of φ, we put edges between the chains corresponding to the variables appearing in C and the vertex v_C corresponding to C in the following way:
  - If C contains the literal u_j, then we take two neighboring vertices v_i, v_{i+1} in the jth chain and add edges from v_i to C and from C to v_{i+1}.
  - If C contains the literal $\overline{u}_j$, then we connect these edges in the opposite direction v_{i+1} to C and C to v_i.

  When adding these edges:
  - We never "reuse" a link v_i, v_{i+1} in a particular chain;
  - Always keep an unused link between every two used links.

  We can do this since every chain has $4m$ vertices, which is more than sufficient for this.

Illustration of the Construction



For every variable u_i we have a "chain" of 4m vertices.

left-to-right traversal = TRUE, right-to-left = FALSE

vertex c_{10} can be visited if chain 1 is traversed left-to-right or if chains 2 or 3 are traversed right-to-left

Correctness of the Reduction

- We show that $\varphi \in \textsc{Sat}$ iff $G \in \textrm{d}\textsc{HamPath}$.

  Suppose that φ has a satisfying assignment $u_1, \ldots, u_n$.

  We exhibit a path that visits all the vertices of G.
  - It starts at v_{start};
  - It travels through all the chains in order;
  - It ends at v_{end}.

  Consider the path that travels the jth chain:
  - Left-to-right, if $u_j = 1$;
  - Right-to-left, if $u_j = 0$.

  This path visits all the vertices except those corresponding to clauses.

  For each clause C, there is at least one literal that is true.

  Use one link on the chain corresponding to that literal to "skip" to the vertex v_C.

  Then continue on as before.

Correctness of the Reduction (Cont'd)

- Suppose that G has an Hamiltonian path P.

  The path P must start in v_{start} and end at v_{end}.

  P needs to traverse all the chains in order.

  Within each chain, it traverses it either left-to-right or right-to-left.

  If it takes the edge $u \to w$, where u is on a chain and w corresponds to a clause, then it must continue with $w \to v$, where v is the vertex adjacent to u in the link.

  Otherwise, the path will get stuck the next time it visits v.

  Define an assignment $u_1, \ldots, u_n$ to φ by setting:

  - $u_j = 1$, if P traverses the jth chain left-to-right;
  - $u_j = 0$, otherwise.

  Then $u_1, \ldots, u_n$ is a satisfying assignment for φ.

Subsection 5

Decision Versus Search

Decision Problems Versus Search Problems

- We have defined NP using Yes/No problems (e.g., "Is the given formula satisfiable?") as opposed to search problems (e.g., "Find a satisfying assignment to this formula if one exists").
- The search problem is obviously harder than the corresponding decision problem.
- So, if $\mathsf{P} \neq \mathsf{NP}$, then neither one can be solved for an NP-complete problem.
- It turns out that for NP-complete problems, decision and search are equivalent in the sense that, if the decision problem can be solved (and, hence, $\mathsf{P} = \mathsf{NP}$), then the search version of any NP problem can also be solved in polynomial time.

Fast Decision and Fast Search Under $\mathrm{P} = \mathrm{NP}$

Theorem

Suppose that $\mathrm{P} = \mathrm{NP}$. Then, for every NP language L and a verifier TM M for L, there is a polynomial-time TM B that, on input $x \in L$, outputs a certificate for x (with respect to the language L and TM M).

- Suppose that $\mathrm{P} = \mathrm{NP}$.

  Let M be a polynomial-time TM.

  Let $p(n)$ be a polynomial.

  We must show that there is a polynomial-time TM B with the following property:

  > For every $x \in \{0,1\}^n$, if there is $u \in \{0,1\}^{p(n)}$, such that $M(x,u) = 1$ (i.e., a certificate that x is in the language verified by M), then $|B(x)| = p(n)$ and $M(x, B(x)) = 1$.

Proof of the Theorem for Sat

- We start by showing the theorem for the case of Sat.

  Let A be an algorithm that decides Sat.

  We come up with an algorithm B that, on input a satisfiable CNF formula φ with n variables, finds a satisfying assignment for φ, using:
  - $2n+1$ calls to A;
  - Some additional polynomial-time computation.

Proof of the Theorem for SAT (Algorithm)

- Let φ be a CNF formula with n variables.

  First, use A to check that the input formula φ is satisfiable.

  If so:

  - First substitute $x_1 = 0$ and, then, $x_1 = 1$ in φ.
    This transformation, which leaves a formula with $n - 1$ variables, can certainly be done in polynomial time.
    Then use A to decide which of the two is satisfiable.
    Say the first is satisfiable. Fix $x_1 = 0$.
  - Continue with the simplified formula, using substitutions for x_2.
  - Continuing this way, we end up fixing all n variables while ensuring that each intermediate formula is satisfiable.

  Thus, the final assignment to the variables satisfies φ.

Proof of the Theorem: The General Case

- Consider an arbitrary NP-language L.

  Recall that the generic reduction from L to $\textsc{Sat}$ is a Levin reduction.

  I.e., it is a polynomial-time computable function f, such that:
  - $x \in L$ iff $f(x) \in \textsc{Sat}$;
  - A satisfying assignment of $f(x)$ is mapped into a certificate for x.

  Therefore, we can:
  - Use the algorithm of the previous slide to come up with an assignment for $f(x)$;
  - Then, map it back into a certificate for x.

Downward Reducibility

- This proof shows that SAT is **downward self-reducible**.
- This means that, given an algorithm that solves SAT on inputs of length smaller than n, we can solve SAT on inputs of length n.
- Using the Cook-Levin reduction, one can show that all NP-complete problems have a similar property.

Subsection 6

coNP, EXP and NEXP

The Class coNP

- Let $L \in \{0,1\}^*$ be a language.
- We denote by $\overline{L}$ the complement of L,

$$\overline{L} = \{0,1\}^* \backslash L.$$

Definition (The Class coNP)

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

- coNP is not the complement of the class NP.
- In fact, coNP and NP have a nonempty intersection, since every language in P is in NP $\cap$ coNP.

  Example: $\overline{\text{SAT}} = \{\varphi : \varphi \text{ is not satisfiable}\} \in \text{coNP}$.

  This holds since, as we know, SAT itself is in NP.

  The satisfying assignment is a polynomial length certificate.

Alternative Definition of coNP and coNP-Completeness

Definition (Alternative Definition of coNP)

For every $L \in \{0,1\}^*$, we say that $L \in$ coNP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M, such that, for every $x \in \{0,1\}^*$,

$$x \in L \quad \text{iff} \quad \forall u \in \{0,1\}^{p(|x|)}(M(x,u) = 1).$$

- A language is coNP-**complete** if:
  - It is in coNP;
  - Every coNP language is polynomial-time Karp reducible to it.

Tautology

- A Boolean formula is called a **tautology** if it is satisfied by every assignment.
- The following language is coNP-complete,
$$\textsc{Tautology} = \{\varphi : \varphi \text{ is a tautology}\}.$$
- $\textsc{Tautology}$ is clearly in coNP.
- We must also show that for every $L \in \text{coNP}$, $L \leq_p \textsc{Tautology}$.

  We just modify the Cook-Levin reduction from $\overline{L}$ to $\textsc{Sat}$.

  For every input $x \in \{0,1\}^*$, that reduction produces a formula φ_x, such that
$$\varphi_x \text{ is satisfiable} \quad \text{iff} \quad x \in \overline{L}.$$
  Consider the formula $\neg\varphi_x$.

  It is in $\textsc{Tautology}$ iff $x \in L$.

P vs. NP, NP vs. coNP and EXP vs. NEXP

- If $\mathsf{P} = \mathsf{NP}$, then $\mathsf{NP} = \mathsf{coNP} = \mathsf{P}$.
- By contraposition, if $\mathsf{NP} \neq \mathsf{coNP}$, then $\mathsf{P} \neq \mathsf{NP}$.
- Most researchers believe that $\mathsf{NP} \neq \mathsf{coNP}$.
- A short certificate that a given formula is a Tautology, i.e., that every assignment satisfies the formula, would be really surprising.
- The class EXP was defined by $\mathsf{EXP} = \bigcup_{c \geq 1} \mathsf{DTIME}(2^{n^c})$.
- This is the exponential-time analog of P.
- The exponential-time analog of NP is the class NEXP, defined by

$$\mathsf{NEXP} = \bigcup_{c \geq 1} \mathsf{NTIME}(2^{n^c}).$$

- Every problem in NP can be solved in exponential time by a brute force search for the certificate.
- So $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$.

P vs. NP and EXP vs. NEXP

Theorem

If EXP $\neq$ NEXP, then P $\neq$ NP.

- We prove the contrapositive, i.e., if P $=$ NP, then EXP $=$ NEXP.

  Suppose $L \in \mathrm{NTIME}(2^{n^c})$ and a NDTM M decides it.

  Consider the language

  $$L_{\mathrm{pad}} = \{\langle x, 1^{2^{|x|^c}} \rangle : x \in L\}.$$

  We claim that L_{pad} is in NP.

  We present an NDTM N for L_{pad}

    Suppose the input is y.

    First check if there is a string z, such that $y = \langle z, 1^{2^{|z|^c}} \rangle$.

    If not, output 0, i.e., halt without going to the state q_{accept}.

    If y is of this form, then simulate M on z for $2^{|z|^c}$ steps.

    Output the answer of M.

P vs. NP and EXP vs. NEXP (Cont'd)

- Clearly, the running time of N is polynomial in $|y|$.

  Hence, $L_{\text{pad}} \in \text{NP}$.

  It follows that, if $\text{P} = \text{NP}$, then L_{pad} is in P.

  But if L_{pad} is in P, then L is in EXP.

  To determine whether an input x is in L, we just:
  - Pad the input;
  - Decide whether it is in L_{pad} using the polynomial-time machine for L_{pad}.