Complexity Theory

- Time Complexity
- Complexity Classes
- Time Hierarchy Theorem
- Polynomial Time P
- Nondeterministic TM
- Nondeterministic Polynomial Time NP
- Polynomial Time Verification
- P vs NP Question
- The Classes EXP and CO-NP

Imdad ullah Khan

Computability: Is the problem solvable by a model (e.g. TM)?

Complexity: Is the problem solvable in $O(n)$ steps?

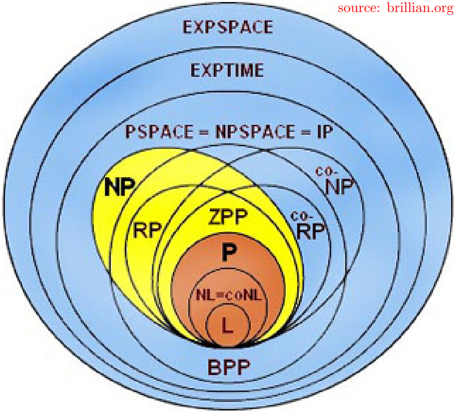
What can be computed with limited *resources*?

Computational resources required by different computation models can be

- Time (number of elementary/bit operations)
- Space (memory cells)
- Random bits (coin flips or calls to pseudorandom number generator by randomized algorithms)
- Communication bandwidth (number of bits transmitted, number of messages exchanged)
- Power or energy (number of KWH consumed, esp. important for battery constrained devices)

Computational Complexity Theory

Structural Complexity attempts to classify computational problems based on the amount of resources required by their solutions



source: brillian.org

Measuring Runtime of a TM

Runtime of a TM is the number of transitions as a function of input size

$\triangleright$ Input size: number of characters on tape

Let M be a TM that halts on all inputs $\triangleright$ $L(M)$ is decidable

Runtime of M is a function $T : \mathbb{N} \mapsto \mathbb{N}$

$T(n)$ is maximum number of transitions performed by M over all inputs of length n

We perform asymptotic analysis on these runtime functions

Measuring Runtime of a TM

Consider the language $L = \{0^k 1^k \mid k \geq 0\}$

Algorithm Check if $w \in L = \{0^k 1^k \mid k \geq 0\}$

if w is not of the form 0^*1^* **then**
    Reject
while both 0's and 1's remain on the tape **do**
    Cross off the first 0 and the first 1 from the tape
if Only 0's or only 1's remain **then**
    Reject
else
    Accept

Measuring Runtime of a TM

Consider the language $L = \{0^k1^k \mid k \geq 0\}$

Algorithm Check if $w \in L = \{0^k1^k \mid k \geq 0\}$, $|w| = n$

---

if w is not of the form 0^*1^* **then** $\triangleright\ O(n)$
    Reject
while both 0's and 1's remain on the tape **do** $\triangleright\ O(n)$
    Cross off the first 0 and the first 1 from the tape $\triangleright\ O(n)$
if Only 0's or only 1's remain **then** $\triangleright\ O(n)$
    Reject
else
    Accept

$$T(n) = O(n) + O(n \times n) + O(n) = O(n^2)$$

Time-Bounded Complexity Classes

Structural Complexity attempts to classify computational problems based on the amount of resources required by their solutions

$TIME(t(n))$

$TIME(t(n)) = \{L' : \exists$ a TM M with runtime $O(t(n))$ and$L(M) = L'\}$

$TIME(t(n))$ is the class of problems decided by a TM in $O(t(n))$ runtime

We just showed that $L = \left\{0^k 1^k \mid k \geq 0\right\} \in TIME(n^2)$

Recall that if $f(n) = O(n)$, then $f(n) = O(n^2)$, $f(n) = O(n^3)$ and so on

We generally want to place problems in the "smallest" class

$\triangleright$ i.e. we want tight upper bounds

Time-Bounded Complexity Classes

$L = \{0^k1^k \mid k \geq 0\} \in TIME(n^2)$ is not the best result

Algorithm Check if $w \in L = \{0^k1^k \mid k \geq 0\}$

if w is not of the form 0^*1^* **then**
  Reject
while both 0's and 1's remain on the tape **do**
  if Number of 0's and number of 1's have different parity **then**
    Reject
  Cross off every other 0 and every other 1 from the tape
if Only 0's or only 1's remain **then**
  Reject
else
  Accept

0000000000000111111111111   x0x0x0x0x0x0xx1x1x1x1x1x1x   xxx0xxx0xxx0xxxx1xxx1xxx1x

xxxxxxx0xxxxxxxxxxxx1xxxxx   xxxxxxxxxxxxxxxxxxxxxxxxxx

$L = \{0^k1^k \mid k \geq 0\} \in TIME(n^2)$ is not the best result

Algorithm Check if $w \in L = \{0^k1^k \mid k \geq 0\}$ $|w| = n$

- **if** w is not of the form 0^*1^* **then** $\triangleright\ O(n)$
  - **Reject**
- **while** both 0's and 1's remain on the tape **do** $\triangleright\ O(\log n)$
  - **if** Number of 0's and number of 1's have different parity **then** $\triangleright\ O(n)$
    - **Reject**
  - Cross off every other 0 and every other 1 from the tape $\triangleright\ O(n)$
- **if** Only 0's or only 1's remain **then**
  - **Reject**
- **else**
  - **Accept**

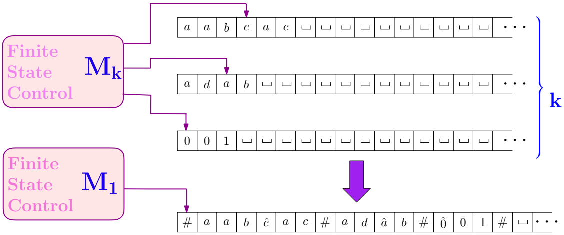
Runtime of this TM is $O(n \log n)$

Multitape TM = Basic TM

A multitape TM has equal computational power as of a basic TM

A basic TM M_2 can simulate any multitape TM M_1

- M_2 stores content of all k tapes in its single tape with $\#$ as separator
  ▷ Assuming $\#$ is not used by M_1
- For each symbol σ (of M_1) M_2 also uses it special version $\hat{\sigma}$. For each section of the tape $\hat{\sigma}$ indicates location of the corresponding head



A basic TM M_2 can simulate any multitape TM M_1

- M_2 stores content of all k tapes in its single tape with $\#$ as separator
  $\triangleright$ Assuming $\#$ is not used by M_1
- For each symbol σ (of M_1) M_2 also uses it special version $\hat{\sigma}$. For each section of the tape $\hat{\sigma}$ indicates location of the corresponding head
- On input $w_1 = w_{11} \dots w_{1\ell}$, $w_2 = w_{21} \dots w_{2m}$, $w_3 = w_{31} \dots w_{3n}$ to M_1
- M_2's tape is $\#\hat{w_{11}} \dots w_{1\ell}\#\hat{w_{21}} \dots w_{2m}\#\hat{w_{31}} \dots w_{3m}\#\sqcup$
- To simulate a transition of M_1, M_2 move its head from first $\#$ to $(k+1)$st $\#$ to find current symbols ($\hat{\sigma}$/virtual heads)
- M_2 then make the transition as dictated by transition of M_2 (writing new symbols and moving all virtual heads)
- If a "head" needs to be moved beyond the $\#$, M_2 first shift all tape content one step to right and continue

Multitape TM can be more efficient

Theorem: A basic TM cannot decide $L = \{0^k 1^k | k \geq 0\}$ in $o(n \log n)$ time

Multitape TM's are not more powerful than basic TM (**for computability)**

Multitape TMs are easier to construct/describe **and also efficient**

We design a 2-tape TM to decide $L = \{0^k 1^k \mid k \geq 0\}$

1. Suppose $w \in \{0, 1\}^*$ is given on tape 1
2. Scan tape 1 left-to-right to check if $w \in 0^* 1^*$ $\triangleright$ $O(n)$
3. Copy all 1's in w from tape 1 to tape 2 $\triangleright$ $O(n)$
4. Scan both tapes left-to-right to see if every 0 on tape 1 has a corresponding 1 on tape 2 and vice-versa, if not **reject** $\triangleright$ $O(n)$
5. **Accept** if heads on both tapes read $\sqcup$

$L = \{0^k 1^k \mid k \geq 0\}$ can be decided in $O(n)$ time by a two-tape TM

Multitape TM can be more efficient

Multitape TMs are easier to construct/describe **and also efficient**

How efficient can multitape TMs be compared to basic TMs?

Theorem: For $B = \{ww|w \in 0,1^*\}$ the gap is quadratic

Theorem: Let $t : \mathbb{N} \mapsto \mathbb{N}$ satisfy $t(n) \geq n$, for all n. Then every $t(n)$ time multi-tape TM M_k, has an equivalent $O(t(n)^2)$ time one-tape TM M_1

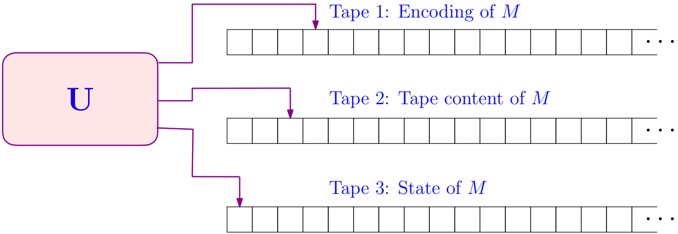
The total length of all tapes of M_k is $\leq t(n)$. To simulate one transition of M_k, M_1 performs at most $O(t(n))$ steps.

Thus, total runtime of M_1 is $O(t(n))^2$

Suppose language A can be decided by a multi-tape TM in $p(n)$ steps, for some polynomial p. Then A can be decided by a one-tape TM in $q(n)$ steps, for some polynomial $q(n)$

Universal Turing Machine

There is a Turing machine U that takes as input an encoding of an arbitrary Turing machine M over Σ and a string $w \in \Sigma^*$ such that U accepts $\langle M, w\rangle$ if and only if M accepts w



The Universal TM with a clock

Theorem

There is a (one-tape) Turing machine U which takes as input:

- *the encoding of an arbitrary TM, M*
- *an input string w*
- *and a string of t 1's with $t > |w|$*

such that

1. $U(\langle M \rangle, w, 1^t)$ *halts in* $O(|M|^2 t^2)$ *steps and*
2. U *accepts* $(\langle M \rangle, w, 1^t) \iff M$ *accepts w in t steps*

Proof Sketch: Make a multi-tape TM M' that takes $(\langle M \rangle, w, 1^t)$ and simulate M on w

M' simulates each step of M in at most $O(t)$ steps

Total runtime of M' is $O(t|M|)$

Simulate M' by a one-tape TM, U with at most quadratic time blow-up

The Time Hierarchy Theorem

Theorem

For all "reasonable" $f, g : \mathbb{N} \mapsto \mathbb{N}$ *and for all* n,

$$f(n) \log f(n) = o(g(n)) \implies TIME(f(n)) \subsetneq TIME(g(n))$$

▷ i.e. with substantial more time, we can solve strictly more problems, as there are languages that can be decided in $O(g(n))$ but not $O(f(n))$

Reasonable means time-constructible $f(n)$ is time-constructible, if $\exists$ a TM, M_f such that $\forall n$, $\exists x, |x| = n$ and $M_f(x)$ halts in exactly $f(n)$ steps.

▷ i.e. $\exists$ M_f, that on input of size n can output $f(n)$ in time $O(f(n))$

Common functions n^i, $n_i(\log n)^j$, 2^{n^i}, $2^{(\log)^i}$ etc. are time-constructible

Why the time-constructible condition? We simulate TM's for a certain time. If a TM runtime is $f(n)$, but the $f(n)$ cannot be computed in time $O(f(n))$, then simulating the TM in $O(f(n))$ time is problematic

The Time Hierarchy Theorem

Theorem

For all "reasonable" $f, g : \mathbb{N} \mapsto \mathbb{N}$ *and for all* n,

$$f(n)\log f(n) = o(g(n)) \implies TIME(f(n)) \subsetneq TIME(g(n))$$

▷ i.e. with substantial more time, we can solve strictly more problems, as there are languages that can be decided in $O(g(n))$ but not $O(f(n))$

To illustrate the ideas, we prove a simpler statement

$Time(n) \subsetneq Time(n^2)$

so we only prove the theorem for the example $f(n) = n$ and $g(n) = n^2$

▷ Note that $n \log n = o(n^2)$

We construct a TM D with runtime $O(n^2)$ whose language $L(D)$ is not accepted by any n time TM, i.e. $L(D) \in TIME(n^2)$ but $L(D) \notin TIME(n)$

The Time Hierarchy Theorem

$Time(n) \subsetneq Time(n^2)$

We construct a TM D with runtime $O(n^2)$ whose language $L(D)$ is not accepted by any n time TM, i.e. $L(D) \in TIME(n^2)$ but $L(D) \notin TIME(n)$

Define a TM D that takes input encoding of a TM M as follows:

$D(\langle M \rangle)$:

Compute $n = |\langle M \rangle|$ (length of $\langle M \rangle$)

Simulate M on $\langle M \rangle$ for $n^{1.9}$ steps and

$$D(\langle M \rangle) = \begin{cases} \textbf{accept} & \text{if } M \text{ rejects} \langle M \rangle \\ \textbf{reject} & \text{if } M \text{ accepts} \langle M \rangle \\ \textbf{accept} & \text{if } M \text{ does not halt (yet) on} \langle M \rangle \end{cases}$$

The Time Hierarchy Theorem

$Time(n) \subsetneq Time(n^2)$

$D(\langle M \rangle)$:

Compute $n = |\langle M \rangle|$ (length of $\langle M \rangle$)

Simulate M on $\langle M \rangle$ for $n^{1.9}$ steps and

$$D(\langle M \rangle) = \begin{cases} \textbf{accept} & \text{if } M \text{ rejects} \langle M \rangle \\ \textbf{reject} & \text{if } M \text{ accepts} \langle M \rangle \\ \textbf{accept} & \text{if } M \text{ does not halt (yet) on} \langle M \rangle \end{cases}$$

Lemma 1: $L(D) \in TIME(n^2)$

Proof: D simulate M for $n^{1.9}$ steps. Maintaining a step a counter and other overhead requires $n^{1.9} \log n$ steps. Thus, $L(D) \in TIME(n^2)$.

The Time Hierarchy Theorem

$Time(n) \subsetneq Time(n^2)$

$D(\langle M \rangle)$:

Compute $n = |\langle M \rangle|$ (length of $\langle M \rangle$)

Simulate M on $\langle M \rangle$ for $n^{1.9}$ steps and

$$D(\langle M \rangle) = \begin{cases} \textbf{accept} & \text{if } M \text{ rejects} \langle M \rangle \\ \textbf{reject} & \text{if } M \text{ accepts} \langle M \rangle \\ \textbf{accept} & \text{if } M \text{ does not halt (yet) on} \langle M \rangle \end{cases}$$

Lemma 1: $L(D) \notin TIME(n)$

Proof: Suppose $L(D) \in TIME(n)$, i.e. $\exists$ a TM R, with runtime $O(n)$ and $L(D) = L(R)$

What is $D(\langle R \rangle)$? D accepts $\langle R \rangle$ if R rejects or loops on $\langle R \rangle$
$\implies L(D) \neq L(R)$

$Time(n) \subsetneq Time(n^2)$

$D(\langle M \rangle)$:

Compute $n = |\langle M \rangle|$ (length of $\langle M \rangle$)

Simulate M on $\langle M \rangle$ for $n^{1.9}$ steps and

$$D(\langle M \rangle) = \begin{cases} \textbf{accept} & \text{if } M \text{ rejects} \langle M \rangle \\ \textbf{reject} & \text{if } M \text{ accepts} \langle M \rangle \\ \textbf{accept} & \text{if } M \text{ does not halt (yet) on} \langle M \rangle \end{cases}$$

Theorem: $L(D) \in TIME(n^2) \setminus TIME(n) \implies Time(n) \subsetneq Time(n^2)$

Technicalities: D can compute $n^{1.9}$, because $f(n)$ is constructible

Need $n^{1.9} > O(n \log n)$, $n = |\langle R \rangle| \geq n_0$ so asymptotic behavior kicks off

$n^{1.9}$ can be replaced with any function in $\omega(n \log n)$ and $o(n^2 \log n)$

Proof of the general theorem follows exactly the same line

The Time Hierarchy Theorem

Theorem

For all "time-constructible" $f, g : \mathbb{N} \mapsto \mathbb{N}$ and for all n,

$$f(n)\log f(n) = o(g(n)) \implies TIME(f(n)) \subsetneq TIME(g(n))$$

$\triangleright$ i.e. with substantial more time, we can solve strictly more problems, as there are languages that can be decided in $O(g(n))$ but not $O(f(n))$

Corollary: $TIME(n) \subsetneq TIME(n^2) \subsetneq TIME(n^3) \subsetneq TIME(n^4)$

Are there important everyday problems that are high up in this time hierarchy?

Is there a natural problem that needs n^{40} time?

Polynomial Time

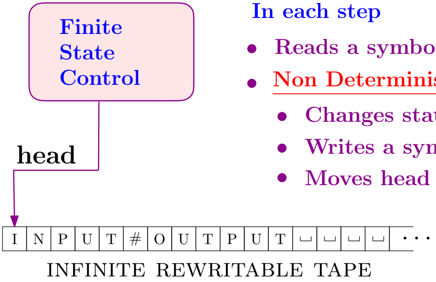
$$P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$$

Extended Church Turing Thesis

Everyone's intuitive notion of efficient algorithms $\subseteq$ Polynmial-Time TM

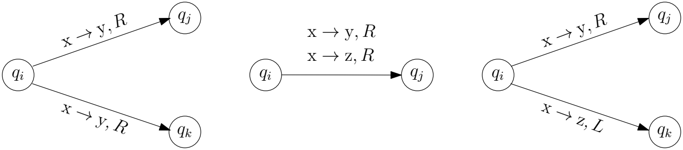
More generally: TM can simulate every "reasonable" model of computation with only polynomial increase in time

A NonDeterministic Turing machine makes nondeterministic choices



In each step

- **Reads a symbol at the head**
- **Non Deterministically**
  - **Changes state**
  - **Writes a symbol at the head**
  - **Moves head to left or right**

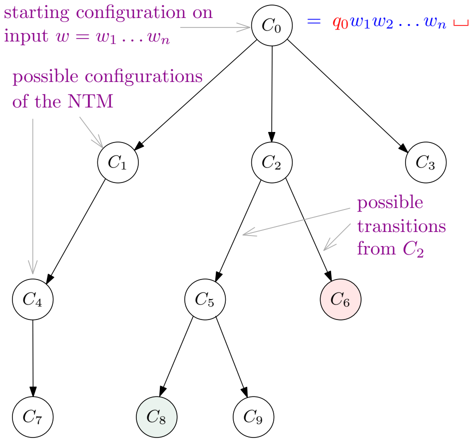


$$\delta : Q \times \Gamma \mapsto \mathcal{P}\left(Q \times \Gamma \times \{L, R\}\right)$$

NonDeterministic Turing Machine

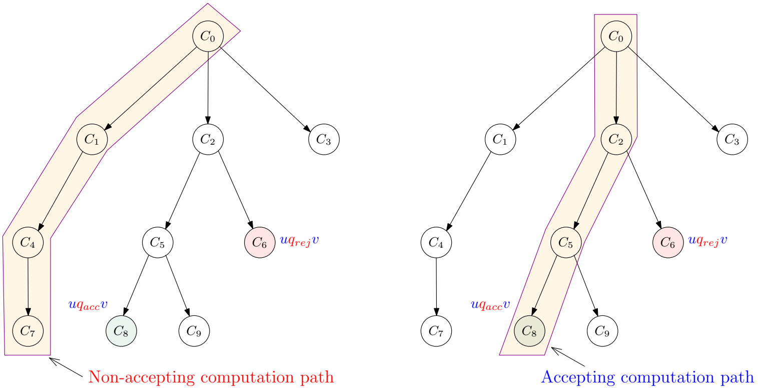
For an NTM a computation is a tree of configurations reachable from the root (starting configuration $qw\sqcup$).

▷ For TM a computation is a sequence (path) of configurations



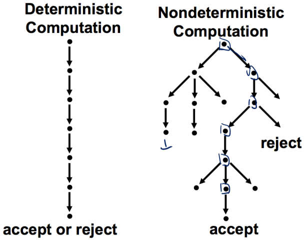
An NTM accepts a string *w* iff some computation path ends in an accepting configuration

i.e. if there is at least one sequence of configurations from the starting configuration to an accepting configuration



Are NFA(NTM) and DFA(TM) equal in computational power?

Ways to think about non-determinism



Parallel computation (with certain restriction) and accepting when one of the node succeeds

Or tree of all possible walks from a start state branching according to symbols on edges and accepting if any leaf node is a final state

Or computing with guessing capability to choose the next state (at certain states) and verifying the right choice

Does verified guessing of NFA (NTM) increases its power over DFA (TM)?

$NTIME(t(n))$

$NTIME(t(n)) = \{L' : \ \exists$ NTM N, with runtime $O(t(n))$ and $L(N) = L'\}$

i.e. $NTIME(t(n))$ is the class of problems decided by a NTM with $O(t(n))$ runtime

$TIME(t(n)) \subseteq NTIME(t(n))$ (because every TM is a NTM)

Nondeterministic Polynomial Time

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

Nondeterministic Time Hierarchy Theorem

Theorem

For all "reasonable" $f, g : \mathbb{N} \mapsto \mathbb{N}$ *and for all* n,

for all n, $f(n)\log f(n) = o(g(n)) \implies NTIME(f(n)) \subsetneq NTIME(g(n))$

The technique for time hierarchy theorem does not directly work.

A NTM running in $O(n)$ time may have $2^{O(n)}$ computation branches.

How to determine in $O(n^2)$ time whether or not it accepts and then flip this answer.

Uses a technique called lazy diagonalization

3-SAT $\in NP$

3-SAT $= \{f : f$ is a satisfiable 3-cnf formula$\}$

$$\text{3-SAT} \in NTIME(n^c) \text{ for constant } c > 1$$

Proof Sketch: Suppose f is input in some natural format and f has n variables and m clauses

1. Check if f is a valid 3-cnf formula
2. Set each variable x_i nondeterministically to 0 or 1
3. Evaluate f with the chosen assignment and **Accept** if f is true

$\textsc{ind-set}(G, k) \in NP$

$\textsc{ind-set}(G, k) = \{G : G$ has an independent set of size $k\}$

$$\textsc{ind-set}(G, k) \in NTIME(n^c) \text{ for constant } c > 1$$

Proof Sketch: Suppose $G = (V, E), |V| = n$ is input in some natural format (e.g. adjacency matrix)

1. Nondeterministically choose a subset $S \subset V$ of size k
2. Verify if S is an independent set, then **Accept**

$\text{HAM-CYCLE}(G) \in NP$

$\text{HAM-CYCLE}(G) = \{G : G \text{ is Hamiltonian}\}$

$$\text{HAM-CYCLE}(G) \in NTIME(n^c) \text{ for constant } c > 1$$

Proof Sketch: Suppose $G = (V, E)$ is input in some natural format (e.g. adjacency matrix)

1. Nondeterministically guess a cyclic permutation of V
2. Verify if the permutation is a Hamiltonian cycle

Polynomial Time Verification

Need to formalize "checking a solution easily" independent of computation

A decision problem X is efficiently verifiable if

1. The claim: "$\mathcal{I}$ is a **Yes** instance of X" can be made in polynomial bits
   - There exists a polynomial sized certificate for **Yes** instances of X
2. A certificate can be verified in polynomial time
   - There exists a polynomial time algorithm $\mathcal{V}$ that takes the instance $\mathcal{I}$ and the certificate $\mathcal{C}$ such that $\mathcal{V}(\mathcal{I}, \mathcal{C}) =$ **Yes** iff $X(\mathcal{I}) =$ **Yes**

Polynomial Time Verification

Computing solution to a problem vs checking a proposed solution

- Sometimes computing and verifying a solution are both "easy"
  - e.g. we can compute a MST of a graph and verify whether a claimed solution is indeed a MST in polynomial time

- Sometimes computing is not easy (yet) but verifying is easy
  - e.g. $3\text{-}\textsc{sat}(f)$ we don't know how to find a satisfying solution (or decide if one exists)
  - But verifying a claimed solution can be done in one scan of f

- Sometimes both computing and verifying a "claim" are not easy
  - e.g. not even clear how to "make" the claim that "G has no Hamiltonian cycle"?

Polynomial Time Verification

The $\textsc{mst}(G, k)$ problem: Is there a spanning tree of G of weight $\leq k$?

$\textsc{mst}(G, k)$ is polynomial time verifiable

- A certificate could be the "*claimed spanning tree*" T for G
  - T can be written by writing vertices ids in some order ▷ $O(n \log n)$ bits
  - Adjacency matrix of edges in T ▷ $O(n^2)$ bits
- A verifier can check
  - if vertices of T are in G
  - If all edges in T are actually from G
  - If sum of weights of edges is k
- Alternatively, a certificate could be an empty string ▷ 0 bits
- A verifier can run Kruskals's algorithm to find a MST T of G
- If $w(T) \leq k$, it verifies the claim otherwise rejects the claim

$3\text{-}\textsc{sat}(f)$ is polynomial time verifiable

- A certificate would be the assignment of 0 and 1's to all variables
- A verifier can evaluate f with the assignment and if the value of f is 1 it outputs **Yes** (=*verified*) otherwise **No** (=*not verified*)

Note that we do not have to design a verifier or a technique for certifying, we only need to prove their existence

- Verifier does not have to be unique
- There can be many ways to certify

  ▷ e.g. an independent set can be certified as the set of vertices, set of edges, complements thereof

- Verifier does not have to read the certificate, recall the requirement $\mathcal{V}(\mathcal{I},\mathcal{C}) =$ **Yes** iff $X(\mathcal{I}) =$ **Yes**

$\textsc{clique}(G, k)$ is polynomial time verifiable

Given an instance $[G, k]$ of $\textsc{clique}(G, k)$

What could be a certificate of claim "$[G, k]$ is **Yes** instance of $\textsc{clique}(\cdot, \cdot)$"?

$\triangleright$ What evidence prove that G has a clique of size k?

Is the certificate of polynomial length?

How can we verify that indeed $[G, k]$ is a **Yes** instance of $\textsc{clique}(G, k)$

$\triangleright$ Does the verifier need to read the certificate?

Is the verifier a polynomial time algorithm?

Polynomial Time Verification

PRIME(n) and COMPOSITE(n) are polynomial time verifiable

▷ Note that they are complement of each other

- A certificate for the COMPOSITE(n) problem can be a factor d
- A verifier can just confirm that $1 < d < n$ and $d|n$

Theorem (AKS(2004))

There exists a polynomial time algorithm to check whether an integer is prime

- A certificate for PRIME(n) can be an empty string
- A verifier exists by the above theorem, using that if n is prime we verify the claim if n is not a prime we reject the claim

Polynomial Time Verification

VERTEX-COVER(G, k) is polynomial time verifiable

- What could be a certificate of claim "G has a vertex cover of size k"?
- How can we verify that indeed "G has a vertex cover of size k?

HAMILTONIAN(G) is polynomial time verifiable

- What could be a certificate of claim "G has a Hamiltonian cycle?"
- How can we verify that indeed G has a Hamiltonian cycle?

Polynomial Time Verification

Are all problems "efficiently" verifiable?

$\overline{\text{3-SAT}}(f)$ It decides whether the given formula f is not satisfiable

▷ sometime referred to as UNSAT(f)

Suppose one wants to claim that the formula f is not satisfiable

▷ Meaning this f is a **Yes** instance of $\overline{\text{3-SAT}}(f)$

How can one make a polynomial sized certificate to make the claim?

▷ "$[0, 1, 1, 0, \dots 1]$ *does not satisfy f", does not mean f is not satisfiable*

The Class P of Problems

The Class P: Decision problems that can be **solved** in polynomial time

- There exists an algorithm that correctly outputs **Yes/No** on any instance
- Recall: polynomial time is a good notion of "reasonable/efficient time"
- Mainly because polynomials are closed under composition (reduction)
- In practice degrees of polynomials are small

(Appropriately defined decision versions of) all these problems are in P

- MST(G, k)
- SHORTEST-PATH(G, s, t, k)
- PRIME(n)
- BIPARTITE-VERTEX-COVER(G, k)
- MAX-FLOW(G, t)

The Class NP of Problems

The Class NP: Decision problems that can be **verified** in polynomial time

A problem X is efficiently verifiable if

- The claim: "$\mathcal{I}$ is a **Yes** instance of X" can be made in polynomial bits
  - There exists a polynomial sized certificate for **Yes** instances of X
- A certificate can be verified in polynomial time
  - There exists a polynomial time algorithm $\mathcal{V}$ that takes the instance $\mathcal{I}$ and the certificate $\mathcal{C}$ such that $\mathcal{V}(\mathcal{I}, \mathcal{C}) =$ **Yes** iff $X(\mathcal{I}) =$ **Yes**

NP stands for "Non-deterministic Polynomial Time"

- 3-SAT(f)
- HAMILTONIAN-CYCLE(G)
- KNAPSACK(U, w, v, C)
- INDEPENDENT-SET(G, k)

Polynomial Time Verification

Theorem

$L \in NP \iff$ *there is a constant k and polynomial-time TM V such that*

$$L = \{\, x \mid \exists y \in \Sigma^* \, [\, |y| \leq |x|^c \text{ AND } V(x, y) \textbf{ accepts} \,] \,\}$$

NP = set of languages L if and only if there is a polynomial-length proofs (aka. certificates or witnesses) for membership in L

Problems with the property that, once you have the solution, it is "easy" to verify the solution

Polynomial Time Verification

Theorem

$L \in NP \iff$ there is a constant k and polynomial-time TM V such that

$$L = \{ x \mid \exists y \in \Sigma^* [\, |y| \leq |x|^c \text{ AND } V(x,y) \textbf{ accepts} \,] \}$$

Proof :

$L = \{ x \mid \exists y \in \Sigma^* [\, |y| \leq |x|^c \text{ AND } V(x,y) \textbf{ accepts} \,] \} \implies L \in NP$

Define NTM $N(x)$: Guess y of length at most $|x|^c$, Output $V(x,y)$

$L \in NP \implies L = \{ x \mid \exists y \in \Sigma^* [\, |y| \leq |x|^c \text{ AND } V(x,y) \textbf{ accepts} \,] \}$

Suppose N is a poly-time NTM that decides L.

Define $V(x,y)$ to accept iff y encodes an accepting computation history of N on x

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

- Let $X \in \mathrm{P}$
- By definition, there exists a polynomial time algorithm $\mathcal{A}$ which decides X
- We need to argue existence of a poly time verifier for X and poly sized certificate for **Yes** instances of X
- The certificate could be an empty string
- Given an instance $\mathcal{I}$ of X and a certificate $\mathcal{C}$ to witness that $X(\mathcal{I}) =$ **Yes**
- $\mathcal{V}$ **can be** $\mathcal{V}(\mathcal{I},\mathcal{C}) := \mathcal{A}(\mathcal{I})$ $\triangleright$ polynomial time
- Essentially ignore the certificate, decide the instance using $\mathcal{A}$ if the output is **Yes** declare verified else not verified
- Notice that the output of this $\mathcal{V}$ is $\mathcal{V}(\mathcal{I}, C) =$ **Yes** iff $\mathcal{A}(\mathcal{I}) =$ **Yes**

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

The following problems we know or can be easily shown to be in P and NP. Notice the corresponding problems are of similar flavor to each other

P	NP
2-SAT	3-SAT
EULER-TOUR	HAMILTONIAN-CYCLE
MST	TSP
SHORTEST-PATH	LONGEST-PATH
INDEPENDENT-SET-TREE	INDEPENDENT-SET
BIPARTITE-MATCHING	3D-MATCHING
BIPARTITE-VERTEX-COVER	VERTEX-COVER
LINEAR PROGRAM	INTEGER LINEAR PROGRAM
PRIME	FACTOR

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

- Many problems in CS, Math, OR, Engineering, etc. are polynomial time verifiable but have no known polynomial time algorithm
- Polynomial time verifiability **seems like** a weaker condition than polynomial time solvability
  - no one has been able to prove that it is weaker (describes a larger class of problems)
- So it is unknown whether $\mathrm{P} = \mathrm{NP}$

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

Many problems in CS, Math, OR, Engineering, etc. are polynomial time verifiable but have no known polynomial time algorithm

Polynomial time verifiability **seems like** a weaker condition than polynomial time solvability

- No proof that it is weaker (i.e. NP describes a larger class of problems)

So it is unknown whether P = NP

Is $\mathrm{P} = \mathrm{NP}$?

The biggest open problem in computer science

Is verifying a candidate solution is easier than solving a problem?

- Majority believes that $\mathrm{P} \neq \mathrm{NP}$
- One can check if any of possible candidate solutions verifies
- But candidate space can be exponential
  - $n!$ possible Hamiltonian cycles are candidates for $\mathrm{TSP}(G, k)$
  - $\binom{n}{k} = O(n^k)$ possible subsets for $\mathrm{CLIQUE}(G, k)$
- **No known " better way" than this**
- **No proof that there is no better way than this**

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

> To say that "P vs NP is the central unsolved problem in computer science" is a comical understatement. P vs NP is one of the deepest questions that human beings have ever asked.
>
> Scott Aaronson

- There is a reason it is one of 7 million-dollar prize problem of the Clay Mathematical Institute (now one of the 6)
- If $\mathrm{P} = \mathrm{NP}$, then mathematical creativity can be automated (the ability to verify a proof would be the same as the ability to find a proof)
- Since verification seems to be way easier, every verifier would have the reasoning power of Gauss and the like
- By just programming your computer in polynomial time you can solve (perhaps) the other 5 Clay Institute problems
- "just because I can appreciate good music, doesn't mean that I would be able to create good music"

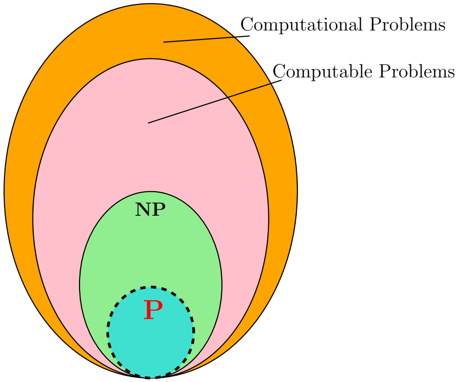
Then why isn't it obvious that $\mathrm{P} \neq \mathrm{NP}$

- Intuition tells us that brute-force search is unavoidable
- It is generally believed that there is no general and significantly better than brute-force method to solve NP problems
- Why can't we prove it?
- It is said that the great physicist Richard Feynman had trouble even being convinced that P vs NP was an open problem
- There are many many problems where we could avoid brute-force search

  ▷ See the list of "hard" problems and their easier "counterparts"
- Though not a decision problem, recall that we discussed that (to impress your boss) you can say that your algorithm for SORTING finds that one unique permutation out of the $n!$ possible ones

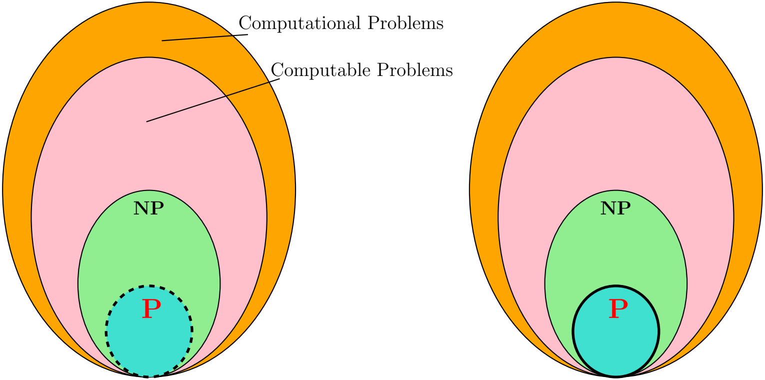
We try to characterize these hard problems and say that almost all of them all essentially the same

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



- For $X \in \mathrm{NP}$ prove that there is no polynomial time algorithm
- You proved $\mathrm{P} \neq \mathrm{NP}$ (You get a million dollars and *A* in this course)

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



- For $X \in \mathrm{NP}$ prove that there is no polynomial time algorithm
- You proved $\mathrm{P} \neq \mathrm{NP}$ (You get a million dollars and *A* in this course)

The Classes P and NP of Problems

The Class P: Decision problems that can be **solved** in polynomial time

The Class NP: Decision problems that can be **verified** in polynomial time

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

The Class CONP of Problems

The Class CONP: Decision problems whose **No instances can be verified** in polynomial time

- Their **No** instances are **Yes** instances of their complement problems
- They are the complements of problems in NP
- Examples: $\overline{\mathrm{SAT}}(f)$, $\overline{\mathrm{HAMILTONIAN}}(G)$
- Note that (the set) CONP is not the complement of NP
- This definition leads to the question is $\mathrm{NP} = \mathrm{CONP}$**?**
- Irrespective of the answer to P vs NP**?** can we certify in polynomial time that G has no Hamiltonian cycle

NP vs coNP

The Class coNP: Decision problems whose **No** instances can be verified in polynomial time

The following result is not very difficult to see

$$\mathrm{P} \subset \mathrm{coNP}$$

Thus

$$\mathrm{P} \subset \mathrm{NP} \cap \mathrm{coNP}$$

We also know that

If $\mathrm{P} = \mathrm{NP}$, then $\mathrm{NP} = \mathrm{coNP}$

This easily follows (read notes) but the converse is not known to be true

It is widely believed that $\mathrm{P} \subsetneq \mathrm{NP} \cap \mathrm{coNP}$

FACTOR(n, k) is in $\mathrm{NP} \cap \mathrm{coNP}$

- FACTOR$(n, k) \in \mathrm{NP}$: A factor $p \leq k$ of n would certify that and can be verified with one division
- FACTOR$(n, k) \in \mathrm{coNP}$: Prime factorization of n can be a certificate that can be verified by checking if "factors" indeed are primes (PRIME$(t) \in \mathrm{P}$)

Is FACTOR$(n, k) \in \mathrm{P}$?

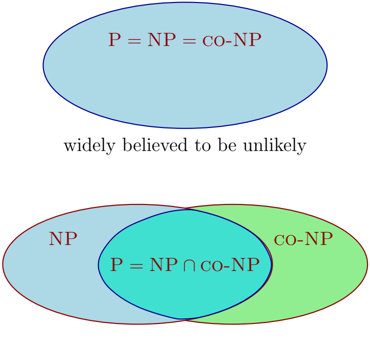
Majority believe it to be not in P, this belief is the basis of RSA cryptosystem

Thus, by this belief $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$

NP = coNP?

The Class coNP: Decision problems whose **No** instances can be verified in polynomial time

Following are possibilities of relationships between these complexity classes



The Class EXP of Problems

The Class EXP: Decision problems that can be **solved** in exponential time

There exists an algorithm that correctly outputs **Yes/No** on any instance and runtime is bounded by an exponential function in size of input

$\mathrm{NP} \subseteq \mathrm{EXP}$ and $\mathrm{coNP} \subseteq \mathrm{EXP}$

- Given that the problem is in NP (coNP), run the polynomial time verification algorithm on all possible certificates
  - there are at most exponentially many certificates
- If on any (all) of the possible certificates we get a **Yes** (**No**) answer from the verifier we get a decision

This gives us the following containment (believed by many to be so)

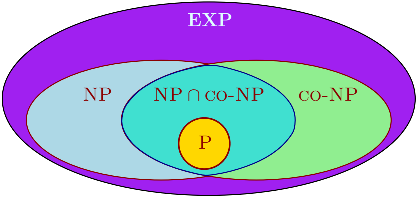


Figure: More likely hierarchy of the discussed complexity classes