The P Versus NP Problem

Gorav Jindal, Anurag Pandey and Harry Zisopoulos

- Hilbert's Entscheidungsproblem asks for a purely mechanical procedure that can decide whether a given mathematical statement is valid or not.
  - What is a "purely mechanical procedure", also called an algorithm?
- Turing(1936) formalized the notion of an algorithm by defining the model of Turing Machines.

Definition 1 (Language)**.** A language L is just a subset of $\{0,1\}^*$, or equivalently a function $L_f : \{0,1\}^* \to \{0,1\}$. Here $L_f(x) = 1$ if and only if $x \in L$.

- Every decidable (computable) language has an associated time complexity.

Definition 2 (Time Complexity)**.** A Turing machine M computes a function $f : \{0,1\}^* \to \{0,1\}$ in time $T(n)$ if $\forall x \in \{0,1\}^*$, M on input x, halts after $\leq T(|x|)$ steps and outputs $f(x)$.

- Time complexity of a language L is the complexity of "fastest" Turing machine computing L_f.
- (Extended) Church Turing Hypothesis :- Turing machines can simulate any computation "efficiently".

Definition 3 (Complexity class)**.** A complexity class is just a set of languages.

- Complexity class P is the set of "easy" languages.

Definition 4 (Complexity class P)**.** A language L is P if time complexity $T(n)$ of L is a polynomially bounded function of n.

- Examples of "easy" problems (languages), i.e, which are in P.
  - Matrix Multiplication, easy to see.
  - Primality testing, a very non-trivial algorithm, so called AKS(2002) algorithm.
  - Linear programming, by so called "ellipsoid method".
- NP is the set of languages whose solutions are "easy to verify".

Definition 5 (Complexity class NP)**.** A language L is NP if there exists a "polynomial time complexity" Turing machine M such that $\forall x \in L, \exists w \in \{0,1\}^{|x|^c}$ with $M(x,w) = 1$, here c is some constant. This w is called a witness/certificate/proof for x, to the fact that $x \in L$. M is called a "verifier" for L. Thus NP is the set of languages which have "small" (polynomial size) witnesses/certificates/proofs and "efficient" (having polynomial time complexity) verifiers.

- See that $\mathsf{P} \subseteq \mathsf{NP}$ because for every language L in P, we can use empty witness and machine for L as the verifier.
- Examples of problems (languages) which are "easy" to verify, i.e, which are in NP.
  - Subset Sum

  - Satisfiability

- Informally, a problem(language) L is NP-hard if a polynomial time algorithm for L implies a polynomial time algorithm for all problems in NP, i.e, L is NP-hard if the statement "$L \in \mathsf{P} \implies \mathsf{P} = \mathsf{NP}$" is true.

- A problem L is NP-complete if L is NP-hard and $L \in \mathsf{NP}$.

- Examples of NP-hard problems (languages).

  - Subset Sum, is NP-complete also.
  - Satisfiability, is NP-complete also.
  - Polynomial System Solving (Hilbert's Nullstellensatz), is NP-complete over finite fields but not known to be NP-complete over infinite fields.
  - Integer Linear Program (ILP), is NP-complete also.
  - Traveling Salesman Problem, is NP-complete also.
  - Tensor Rank, is not known to be NP-complete.

Problem 6 (P vs NP Problem)**.** Is P a strict subset of NP or is $\mathsf{P} = \mathsf{NP}$?

Note that if any NP-complete problem belongs to P, then all NP-complete problems belong to P. That would imply $\mathsf{P} = \mathsf{NP}$. So "P vs NP Problem" essentially asks whether any one of the NP-complete problems/languages is in P. We now know thousands of NP-complete problems. It is a common theme that every area of mathematics has corresponding computational problems which are NP-complete. Sometimes these problems are only known to be NP-hard and not NP-complete, as Polynomial System Solving. Gödel essentially observed in his letter to von Neumann that the following language L_{math} is in NP. We use the notation $1^n = \underbrace{111\ldots1111}_{n\text{-times}}$.

$$L_{\text{math}} \stackrel{\text{def}}{=\!=} \{(\phi, 1^n) \mid \phi \text{ is a valid mathematical statement having a proof of length at most } n\}.$$

Note that proof of ϕ of length at most n can serve as the witness/certificate for checking if $(\phi, 1^n) \in L_{\text{math}}$, this checking can be done in polynomial time. Thus $L_{\text{math}} \in \mathsf{NP}$. Gödel also asks in that letter to von Neumann, whether or not $L_{\text{math}} \in \mathsf{P}$?

$\mathsf{P} = \mathsf{NP}$ would imply that $L_{\text{math}} \in \mathsf{P}$. This would imply that L_{math} can be solved in time $O(n^c)$ for some $c \geq 0$. Assuming that this c is not too large, would imply the existence of a "practical" algorithm for L_{math}. That would imply that there exists an efficient algorithm which can check whether there exists a "short" proof of a given mathematical statement, thus undermining the creative effort of a mathematician in finding "short" proofs of mathematical statements.

Theorem 7 (Ladner's theorem)**.** *If $\mathsf{P} \neq \mathsf{NP}$ then there exist languages $L \in \mathsf{NP} \setminus \mathsf{P}$ which are not NP-complete.*