1 Importance of the Cook-Levin Theorem

There is a trivial NP-complete language:

$$L_u = \{(M, x, 1^k) \mid \text{NTM } M \text{ accepts } x \text{ in} \leq k \text{ steps}\}$$

Exercise: Show that L_u is NP-complete.

The language L_u is not particularly interesting, whereas SAT is extremely interesting since it's a well-known and well-studied natural problem in logic. After Cook and Levin showed NP-completeness of SAT, literally hundreds of other important and natural problems were also shown to be NP-complete. It is this abundance of natural complete problems which makes the notion of NP-completeness so important, and the "P vs. NP" question so fundamental.

2 Viewing NP as a game

Nondeterministic computation can be viewed as a two-person game. The players are Prover and Verifier. Both get the same input, e.g., a propositional formula ϕ. Prover is all-powerful (but not trust-worthy). He is trying to convince Verifier that the input is in the language (e.g., that ϕ is satisfiable). Prover sends his argument (as a binary string) to Verifier. Verifier is computationally bounded algorithm. In case of NP, Verifier is a deterministic polytime algorithm.

It is not hard to argue that the class NP of languages L is exactly the class of languages for which there is a pair (Prover, Verifier) with the property: For inputs in the language, Prover convinces Verifier to accept; for inputs not in the language, any string sent by Prover will be rejected by Verifier. Moreover, the string that Prover needs to send is of length polynomial in the size of the input.

Once we have this set-up, we can vary computational resources allowed for Verifier to use. For instance, we can let Verifier be a randomized polytime algorithm, thus allowing some small probability that Verifier is incorrectly convinced the input outside the language. We can also allow Verifier random access to the string sent by Prover, i.e., Verifier can choose a few (say, 3) locations in the string and look only at the corresponding 3 bits. What class of languages can we capture with this kind of randomized protocols where Verifier reads only 3 bits of Prover's string? It turns out all of NP! This is basically the content of the famous PCP Theorem (where "PCP" stands for "Probabilistically Checkable Proofs"). That is, for every language in NP, there is a pair (Prover, Verifier) such that, for inputs in the language Prover sends a polynomial-length string that is accepted by Verifier with probability 1, while for inputs not in the language any such string is rejected by Verifier with constant probability

[1] This lecture is a modification of notes by Valentine Kabanets

(say, 1/2). Moreover, Verifier always reads (randomly selected) constantly many (say, 3) bits of Prover's string.

We will talk more about this alternative characterization of NP and its applications later in the course.

3 Co-NP

We say that a language $L \in \mathsf{coNP}$ if the complement of L is in NP.

In other words, if $L \in \mathsf{coNP}$, then there is a NTM M with the property: if $x \in L$, then *every* computation path of M on x is accepting; if $x \notin L$, then at least one computation path of M on x is rejecting.

Another, equivalent definition of coNP is as follows. A language $L \in \mathsf{coNP}$ if there exist a constant c and a polynomial-time computable relation $R \in \mathsf{P}$ such that

$$L = \{x \mid \forall y, |y| \leq |x|^c, R(x, y)\}$$

Open Question: $\mathsf{NP} \stackrel{?}{=} \mathsf{coNP}$

Define the language

$$TAUT = \{\phi \mid \phi \text{ is a tautology (i.e., identically true)}\}$$

Theorem 1. *TAUT is* coNP*-complete.*

Proof. The proof is easy once you realize that TAUT is essentially the same as the complement of SAT. Since SAT is NP-complete, its complement is coNP-complete. (Check this!) □

The "NP vs. coNP" question is about the existence of short proofs that a given formula is a tautology.

The common belief is that $\mathsf{NP} \neq \mathsf{coNP}$, but it is believed less strongly than that $\mathsf{P} \neq \mathsf{NP}$.

Lemma 2. *If* $\mathsf{P} = \mathsf{NP}$*, then* $\mathsf{NP} = \mathsf{coNP}$*.*

Proof. First observe that $\mathsf{P} = \mathsf{coP}$ (check it!) Now we have $\mathsf{NP} = \mathsf{P}$ implies $\mathsf{coNP} = \mathsf{coP}$. We know that $\mathsf{coP} = \mathsf{P}$, and that $\mathsf{P} = \mathsf{NP}$. Putting all this together yields $\mathsf{coNP} = \mathsf{NP}$. □

The contrapositive of this lemma says that: $\mathsf{NP} \neq \mathsf{coNP}$ implies $\mathsf{P} \neq \mathsf{NP}$.

Thus, to prove $\mathsf{P} \neq \mathsf{NP}$, it is enough to prove that $\mathsf{NP} \neq \mathsf{coNP}$. This means that resolving the "NP vs. coNP" question is probably even harder than resolving the "P vs. NP" question.