INFR11102: Computational Complexity 17/10/2019

Lecture 10: NP-intermediate candidates

Lecturer: Heng Guo

1 The class coNP

Recall that by $\overline{L}$ we denote the complement language of L, namely $x \in \overline{L}$ if and only if $x \notin L$. Define

$$\mathtt{coNP} := \{L \mid \overline{L} \in \mathtt{NP}\}. \tag{1}$$

Recall the verification definition of NP. The class coNP then is the class of problems whose "NO" instances have succinct certificates.

Similar to NP, there is an alternative definition of coNP to (1).

Definition 1. *A language L is in* coNP *if there is a polynomial $p(\cdot)$ and a polynomial-time TM M such that*

$$x \in L \Leftrightarrow \forall y, \;\; |y| \le p(|x|), \;\; M(x,y) = 1.$$

Recall that in the alternative definition of NP, a certificate y is required to exist so that (x,y) is accepted by the verifier M. Instead, in Definition 1, we require that all certificates are accepted.

Similar to the NP-completeness of Sat, the most commonly used complete language (with respect to Karp reductions) of coNP is "tautology":

Name: Taut

Input: A CNF formula φ.

Output: Is φ always true?

Essentially, coNP is characterised by the universal quantifier. For example, a "YES" instance of Taut is a formula φ such that for all assignments σ of the variables, φ evaluated under σ is always true.

Another popular conjecture is that $\mathtt{NP} \neq \mathtt{coNP}$, but again, this is still an open problem.

2 NP-intermediate problems

Due to Ladner's theorem, there is an infinite hierarchy of problems of increasing complexity between P and NP-complete, if $\mathtt{P} \neq \mathtt{NP}$. However, no natural intermediate problem is found yet. A couple of potential candidates are Factoring and GI (graph isomorphisms).

2.1 PRIMES is in $\mathtt{NP} \cap \mathtt{coNP}$

Before talking about Factoring, we will start with primality testing.

Name: PRIMES

Input: A natural number n in binary.

Output: Is n a prime number?

The naive algorithm to solve PRIMES is to enumerate all possible prime factors, from 1 to at most $\sqrt{n}$. The problem with this approach is that the length of n in binary is only $\log n$. Thus doing $\sqrt{n}$ operations is exponential in terms of the input size.

It is rather clear that PRIMES $\in \mathtt{coNP}$. The reason is that, if n is not a prime number, then there must be a prime p such that $1 < p < n$ and $p \mid n$. This is a polynomial-size certificate for "NO" instances of PRIMES.

What is perhaps surprising is that PRIMES is also in $\mathtt{NP}$. This was proved by Vaughan Pratt in 1975 [Pra75].

Theorem 1 (Pratt's theorem)**.** *PRIMES $\in \mathtt{NP} \cap \mathtt{coNP}$.*

A well-known theorem in elementary number theory is Fermat's little theorem.

Theorem 2 (Fermat's little theorem)**.** *If p is a prime, then for any integer a, $a^p \equiv a \mod p$. In particular, if $p \nmid a$, $a^{p-1} \equiv 1 \mod p$.*

Fermat's little theorem in fact has a converse, known as Lehmer's theorem [Leh27].

Theorem 3 (Lehmer's theorem)**.** *Let n be an integer. If there is an integer a such that*

1. *$a^{n-1} \equiv 1 \mod n$, and*
2. *For every prime factor q of $n-1$, it is not the case that $a^{(n-1)/q} \equiv 1 \mod n$,*

then n is a prime.

Theorem 3 is a simple fact about cyclic groups.

If n is a prime, then such an a always exists. In fact, any $1 < a < n$ works. Clearly $a < n$ is a succinct certificate. Does this show Theorem 1?

Not yet! The problem is that once we have guessed a, it is easy to check Condition 1 in Theorem 3. However we still need to check Condition 2 in Theorem 3. (For example, $(n-1)^{n-1} \equiv 1 \mod n$ always holds if n is odd.) To do this, we need to know a prime factorisation of $n-1$. We can then guess a factorisation of $n-1$, but how do we make sure all factors are prime? We recurse. Namely, we will guess a along with a factorisation $p_1 p_2 \cdots p_k = n-1$ (not necessarily distinct primes), and then guess certificates for the primality of $p_1, p_2, \cdots, p_k$ recursively.

The key question is how large will this certificate get?

We claim that there are at most $4 \log_2 n - 4$ numbers encountered in this recursion. This holds for the base case of $n = 3$. For $n > 3$, let $\prod_{i=1}^{k} p_i = n-1$ where $p_1, p_2, \cdots, p_k$ are (not necessarily distinct) primes and $k \geq 2$. Then for n, by the induction hypothesis, the total amount of numbers encountered in the recursion is at most

$$\begin{aligned} 1 + \sum_{i=1}^{k} (4 \log_2 p_i - 4) &= 1 - 4k + 4 \log_2 \prod_{i=1}^{k} p_i \\ &= 1 - 4k + 4 \log_2 (n-1) \\ &\leq 4 \log_2 n - 4, \end{aligned}$$

since $k \geq 2$ and $\prod_{i=1}^{k} p_i = n - 1$. Clearly all these numbers are at most n, and thus use at most $\log_2 n$ bits. The total size of the certificate is $O((\log_2 n)^2)$.

A consequence of Theorem 1 is that, if PRIMES is $\mathtt{NP}$-complete, then $\mathtt{NP} = \mathtt{coNP}$, which contradicts to many people's beliefs. The possibility that PRIMES is $\mathtt{NP}$-complete was further reduced, when efficient *randomised* primality testing algorithms are found by Gary Miller [Mil76] and Michael Rabin [Rab80]. Note that we do not know a randomised polynomial-time algorithm for SAT. We will cover randomised computation later on. The next surprising fact is that, in fact, PRIMES is in $\mathtt{P}$! This was shown by Agrawal, Kayal and Saxena [AKS04] through a complicated derandomisation technique.

2.2 Factoring

A related problem, that is still not known to be in $\mathtt{P}$ or not, is FACTORING.

Name: FACTORING

Input: An integer n in binary.

Output: A prime factorisation of n.

Apparently if we formalise it this way, then it is not a decision problem. Nonetheless, this problem has very important applications in cryptography, since a polynomial-time algorithm for FACTORING would crack the RSA public-key system. Although a polynomial-time (deterministic or randomised) algorithm is not known, an efficient quantum one is! This is the famous Shor's algorithm [Sho97].

On the other hand, it would be very surprising if FACTORING is $\mathtt{NP}$-hard. This is because its decision version is in $\mathtt{NP} \cap \mathtt{coNP}$.

Name: FACTOR

Input: Integers n and k in binary.

Output: Does n have a prime factor that is at most k?

Clearly FACTOR $\in \mathtt{NP}$ since a prime $p \leq k$ such that $p \mid n$ is a succinct certificate. On the other hand, since we know PRIMES is in $\mathtt{P}$, we can verify a prime factorisation of n. The prime factorisation is thus a certificate that n does not have a prime factor that is at most k. Thus, FACTOR $\in \mathtt{NP} \cap \mathtt{coNP}$.

Therefore, if FACTORING or FACTOR was $\mathtt{NP}$-hard, then $\mathtt{NP} = \mathtt{coNP}$, which is a surprising consequence.

2.3 Graph Isomorphisms

Another candidate intermediate problem is graph isomorphism (GI). This is a classical computational problem, and its complexity is yet unsettled.

Name: GI

Input: Two graphs G_1 and G_2.

Output: Are G_1 and G_2 isomorphic?

At first glance, GI may look not that difficult (is it?). Essentially, the only trouble is that the vertices may come in different orderings in G_1 and G_2, and "all" we need to figure out is whether there is a reordering or not to make them identical. However, after decades of research, there is no known polynomial-time algorithm yet.

It is straightforward that GI $\in$ $\mathtt{NP}$. The certificate is simply a permutation of all vertices so that G_1 and G_2 are identical. However, there is no clear way to show that GI $\in$ $\mathtt{coNP}$. Thus we do not know the kind of evidence such that FACTOR is not $\mathtt{NP}$-hard. In fact, we do have evidence that GI is not likely to be $\mathtt{NP}$-hard. To talk about this evidence, we will need to go beyond $\mathtt{NP}$, to the so-called "polynomial hierarchy".

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[Leh27] Derrick H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bull. Amer. Math. Soc.*, 33(3):327–340, 05 1927.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976.

[Pra75] Vaughan R. Pratt. Every prime has a succinct certificate. *SIAM J. Comput.*, 4(3):214–220, 1975.

[Rab80] Michael O. Rabin. Probabilistic algorithm for testing primality. *J. Number Theory*, 12(1):128 – 138, 1980.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.