E0 224: Computational Complexity Theory

Chandan Saha

Lecture 7

Abhijat Sharma
Indian Institute of Science
Bangalore, India

27 August 2014

1 Class $co - \mathbb{NP}$

If $L \subseteq \{0,1\}^*$ is any language, we denote the *complement* of L by $\bar{L}$. That is, $\bar{L} = \{0,1\}^* \setminus L$. Hence, we can define:

Definition 1. *The complexity class $co - \mathbb{NP}$ consists of all the languages L, such that $\bar{L} \in \mathbb{NP}$. For example,*

$$\overline{SAT} = \{\phi \colon \phi \quad \textit{is not satisfiable} \quad \}$$

It can also be equivalently defined as follows:

Definition 2. *A language L belongs to the complexity class $co - \mathbb{NP}$ if there exists a polynomial function $q()$, and a deterministic polynomial-time TM M, such that*

$$x \in L \quad \textit{iff} \quad \forall u \in \{0,1\}^{q(x)}, M(< x, u >) = 1.$$

One can also define $co-\mathbb{NP}-completeness$ in a way similar to $\mathbb{NP}-completeness$ under polynomial-time reduction:

Definition 3. *A language L is said to be $co - \mathbb{NP} - complete$ iff*

1. *$L \in co - \mathbb{NP}$*
2. *Every other language $L' \in co - \mathbb{NP}$ is polynomial-time Karp reducible to L*

For example, $\overline{SAT} \in co - \mathbb{NP} - complete$. Also, observe the following language:

$$TAUTOLOGY = \{\phi \colon \phi \quad \text{is true for all possible assignments} \quad \}$$

It can be easily verified that $TAUTOLOGY \in co - \mathbb{NP} - complete$.

NOTE Therefore, similar to $co - \mathbb{NP}$, we can define a complexity class co-C for every class C.

$$co - C = \{L \colon \bar{L} \in C\}$$

From this definition, it is easy to observe that $\mathbb{P} = co - \mathbb{P}$.

But, it is important to realise the difference between the classes $\bar{C}$ and $co-C$. This can be easily seen by the following statement:

$$\text{If there are 2 classes} \quad C_1, C_2 \quad \text{such that} \quad C_1 \subseteq C_2, \quad \text{then} \quad co-C_1 \subseteq co-C_2$$

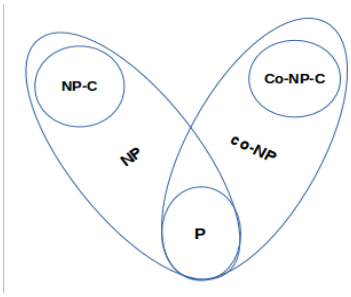
The above statement. we know, is definitely not true for $\bar{C_1}$ and $\bar{C_2}$.

1.1 Is $\mathbb{NP} = co-\mathbb{NP}$?

Claim If $\mathbb{NP} \neq co-\mathbb{NP}$ then $\mathbb{P} \neq \mathbb{NP}$.

Proof Suppose $\mathbb{P} = \mathbb{NP}$
$\Longrightarrow$co-P=co-NP
$\Longrightarrow$P=co-NP($\because \mathbb{P} = co-\mathbb{P}$)
$\Longrightarrow$NP=co-NP
Hence, we arrive at a contradiction.

Therefore, the assumption that $\mathbb{NP} \neq co-\mathbb{NP}$ is a stronger assumption than $\mathbb{P} \neq \mathbb{NP}$. Thus, the overall picture looks somewhat like this:



Observe that $\mathbb{P} \subseteq \mathbb{NP} \cap co-\mathbb{NP}$. But is $\mathbb{P} = \mathbb{NP} \cap co-\mathbb{NP}$?

1.2 Significance of the above questions

Open questions such as whether $\mathbb{P} = \mathbb{NP}$ or $\mathbb{NP} = co-\mathbb{NP}$ or if $\mathbb{P} = \mathbb{NP} \cap co-\mathbb{NP}$ are known as *lower bound* questions in complexity theory. They form the basis of large volumes of further research and many significant results in the area. It can be interesting to think about the consequences of solving any of these open problems. For example, what would happen if $\mathbb{P} = \mathbb{NP}$? For every hard problem that could only be efficiently verified, we would be able to find the correct answer in polynomial time. It might imply that mathematicians could be replaced by efficient theorem-discovering machines, Artificial Intelligence would be perfect and a lot of popular encryption schemes would lose their effects.

Similarly, if $\mathbb{NP} = co-\mathbb{NP}$, we would need to find efficient certificates to verify statements for which essentially no certificate exists. Thus, to sum up, the consequences would be fairly dramatic and the world would reduce to a computational Utopia. [1]

We will now discuss one of the major techniques used to prove some of such lower bound results.

2 Diagonalisation

Definition 4. *Any proof technique is called a digonalisation technique if it solely uses the following features:*

1. *Every string $x \in \{0,1\}^*$ represents some TM, denoted by M_x, and every TM can be represented by infinitely many strings.*
2. *There exists a Universal TM that, given any string x, can simulate the execution of the machine M_x without much overhead.*

There are a large number of lower bound results that are proved using the diagonalisation technique described above. We also witnessed, in an earlier lecture, how this technique was used to prove the undecidability of the Halting Problem. Some more such problems are are:

1. Deterministic Time Hierarchy Theorem
2. Non-Deterministic Time Hierarchy Theorem
3. Ladner's Theorem
4. Baker, Gill, Solovay Theorem

We now describe some of the above proofs in detail.

3 Deterministic Time Hierarchy

Definition 5. *We say a function $f\colon \mathbb{N} \mapsto \mathbb{N}$ is time-constructible if $f(n)$ can be computed from n in $O(f(n))$ time.*

Most of the functions we usually deal with in day-to-day life are all time-constructible functions. For example, all polynomial functions, logarithmic functions, trigonometric functions, exponential functions etc. Now, we define the time hierarchy theorem with respect to such functions.

Theorem 1. *Suppose $f, g\colon \mathbb{N} \mapsto \mathbb{N}$ are two time-constructible functions, such that $g(n) = \omega(f(n)log(f(n)))$, then*

$$DTIME(f(n)) \subsetneq DTIME(g(n))$$

Proof: Let us first prove a simple result, DTIME(n)$\subsetneq DTIME(n^2)$, which can be later generalised to the above theorem.
Goal: To define a language L$\subseteq \{0,1\}^*$ such that $L \in DTIME(n^2)$ but $L \notin DTIME(n)$.

On input x, the universal TM U simulates M_x on x for exactly $\mid x \mid^2$ steps. If M_x halts and outputs b, then U outputs $1 - b$. Otherwise, U outputs 0.
We say that L is the language decided by this machine U.
Observe that $L \in DTIME(n^2)$ as the machine U accepts L in at-most $O(n^2)$ time steps.

Now, we wish to show that there's no deterministic TM that can decide L in linear time steps.
For contradiction, suppose N is a TM whose running time is bounded by $C_N n$, where C_N is a constant and N decides the language L.
How long does it take for the universal TM to simulate N on input of length n? As we know, the universal TM only adds a logarithmic overhead to the runtime, it would take $C_U C_N nlogn$ time for this simulation. And, asymptotically we know that $n^2 \ggg C_U C_N nlogn$ for $n > n_0$.

Claim There exist a binary representation α of N such that $\mid \alpha \mid > n_0$, and the TM N and the TM U disagree on the input α.

Proof On input α, the universal TM U simulates $M_\alpha = N$, for exactly $\mid \alpha \mid^2$ steps of U. (Note that $\mid \alpha \mid^2 \ggg C_U C_N \mid \alpha \mid log \mid \alpha \mid)$. This means that N definitely halts during this simulation, as running time of N is bounded by $C_N \mid \alpha \mid$. Now, U outputs exactly opposite to the output of N.
Thus, if N and U disagree on the input α, and U decides L, then N cannot decide L. Hence, it is proved that $L \in DTIME(n^2)$ but $L \notin DTIME(n)$.
Now, this proof would also hold for any other $g(n)$ such that $f(n) = \omega(nlogn)$, and hence the aboove theorem can be verified by replacing n by any time-constructible function $f(n)$

Consequences The time-hierarchy theorem guarantees that all the inequalities of DTIME complexity classes are all proper inequalities, such as:

$$\mathbb{P} \subsetneq DTIME(2^n) \subsetneq DTIME(2^{2n})...$$

These hierarchies eventually give rise to deterministic and non-deterministic versions of exponentials hierarchies [2]. The theorem also guarantees that there are problems in $\mathbb{P}$ requiring arbitrary large exponents to solve; in other words, $\mathbb{P}$ does not collapse to $DTIME(n^k)$ for any fixed k.

4 Ladner's Theorem

Theorem 2. *If $\mathbb{P} \neq \mathbb{NP}$, then there exists an "intermediate" $\mathbb{NP}$ language $L \subseteq \{0,1\}^*$, that is neither in $\mathbb{P}$ nor is $\mathbb{NP}$-complete.*

Proof For every function $H : \mathbb{N} \mapsto \mathbb{N}$, we define

$$SAT_H = \{\phi \circ 1^{n^{H(n)}} : \phi \in SAT \quad \text{and} \quad \mid \phi \mid = n\}$$

Observe that the value of $H(n)$ on n, only decides membership of strings of length greater than n, in SAT_H.

n	H(n)	SAT_H
1	H(1)	strings of length 1 in SAT_H
2	H(2)	strings of length 2 in SAT_H
3	H(3)	strings of length 3 in SAT_H
.	.	.
.	.	.
.	.	.
∞	.	.

Now, we try to define the function $H(n)$:

Definition 6. *$H(n)$ is the smallest $\alpha < loglogn$ such that, for every $x \in \{0,1\}^*$, and $\mid x \mid \leq logn$, M_α outputs $SAT_H(x)$ in time $\alpha \mid x \mid^\alpha$. If such an α doesn't exist, then $H(n) = loglogn$.*

Fact 1 $H(n)$ has been defined in such a way that if $SAT_H \in \mathbb{P}$, then $H(n) = O(1)$. And, if $H(n) < C$ for some constant C, for infinitely many values of n, then $SAT_H \in \mathbb{P}$.
This fact implies that If $SAT_H \notin \mathbb{P}$, then $H(n) \longrightarrow \infty$ as $n \longrightarrow \infty$.

Proof of one-side If $SAT_H \in \mathbb{P}$, then $H(n) = O(1)$.
Let the polynomial time TM be N, whose running time is $C \mid x \mid^d$. Pick a string representation of N (say α), that is larger than both C and d.
Then we claim that $H(n) \leq \alpha$ for all values of n. This can be observed from the definition of H, which implies that for $n > 2^{2^i}$, $H(n) \leq i$. So, as the value of n grows larger and larger, $H(n)$ grows approximately close to a small constant value.

Proof of the other side If $H(n) \leq C$ infinitely often, then $SAT_H \in \mathbb{P}$.
Suppose there exists an x, such that $H(n) = x$ for infinitely many n's. This implies that the TM M_x decides SAT_i in xn^x time, because if this is not the case, if any M_i does not decide SAT_i within this bound, $H(n) \neq i$ for every $n > 2^{|x|}$. Hence, $SAT_H \in \mathbb{P}$.

Fact 2 $H(n)$ is computable in a polynomial time function of n.

It can be easily verified that $SAT_H \in \mathbb{NP}$. The rest of this proof will proceed under the assumption that $\mathbb{P} \neq \mathbb{NP}$.

Claim 1 $SAT_H \notin \mathbb{P}$

Proof Suppose $SAT_H \in \mathbb{P}$, then let A be the algorithm that decides SAT_H in polynomial time. We show that this would imply that $SAT \in \mathbb{P}$, by giving another polynomial time algorithm $A\prime$ that decides SAT.
The algorithm $A\prime$, when given as input any boolean function ϕ, calculates the length of ϕ, say n and computes $H(n)$ in polynomial time. Finally, it gives the string $\phi \circ 1^{n^{H(n)}}$ as an input to the algorithm A.
As we know that no such polynomial time algorithm can exist for an NP-complete language such as SAT, we arrive at a contradiction.

Claim 2 SAT_H is not $\mathbb{NP} - complete$

Proof Suppose SAT_H is $\mathbb{NP} - complete$, then we must have a polynomial time reduction from SAT to SAT_H. ($SAT \in \mathbb{NP}$ and every problem in $\mathbb{NP}$ can be reduced to an $\mathbb{NP} - complete$ problem)
Hence, we have

$$\phi \xrightarrow[\text{poly-time reduction}]{f} \psi \circ 1^{n^{H(n)}}, \mid \phi \mid = m \quad \text{and} \quad \mid \psi \mid = n$$

such that

$$\phi \quad \text{is satisfiable iff } \psi \quad \text{is satisfiable}$$

Therefore, $\mid \psi \circ 1^{n^{H(n)}} \mid = n + 1 + n^{H(n)} \leq p(m)$ where $p(m)$ is a polynomial function in m. Because $SAT_H \notin \mathbb{P}$, $H(n)$ tends to a very large value for large n.
Let's assume that $H(n) > 10$ and $p(m) = m^5$, then $n + 1 + n^{H(n)} \leq m^5 \Rightarrow n \leq \sqrt{m}$

Thus, the larger the values of $H(n)$, and proportional to m (the length of the formula ϕ), the mapping will reduce ϕ to ψ where the length of ψ would be smaller by some fixed polynomial factor ($\sqrt{m}$ in the above example). Thus, performing the same reduction againon ψ and so on, we will get a simple polynomial time algorithm that would decide SAT, which we know cannot be possible. Hence, we arrive at another contradiction.

By Claims 1 and 2, we observe that SAT_H is a language in $\mathbb{NP}$ that is neither in $\mathbb{P}$ nor is $\mathbb{NP} - complete$.

Consequences There are very few problems that fall into this "intermediate" class between $\mathbb{P}$ and $\mathbb{NP} - complete$, since the status of most natural languages has been resolved thanks to clever algorithms or reductions. Some of the exceptions are the problems of factoring, graph isomorphism, discrete log etc, for which there is no proof yet that they are $\mathbb{NP} - complete$ or not.

5 Supplementary Exercises

The following problems can be taken as reading exercises for better understanding of the above mentioned topics covered in this lecture: (Most of these have also been included as Assignment-1 problems)

1. Explore whether $PRIMES \in co - \mathbb{NP}$ or $PRIMES \in \mathbb{NP} \cap co - \mathbb{NP}$ [3].
2. Non-deterministic Time Hierarchy Theorem, given initially by Stephen Cook in 1972.
3. Find examples of functions that are not Time-constructible.
4. Fact 2 about the function H(n), mentioned while proving the Ladner's Theorem.

References

[1] Sanjeev Arora and Boaz Barak, 2007. *Computational Complexity: A Modern Approach*, Cambridge University Press.

[2] Time Hierarchy Theorem. *http://en.wikipedia.org/wiki/Time_hierarchy_theorem*, Wikipedia.

[3] Vaughan Pratt. *Every prime has a succinct certificate*, SIAM Journal on Computing, vol.4.