CSCE-637 Complexity Theory

Lecture #2, October 28, 2020

Lecturer: Professor Jianer Chen

3 The Karp-Lipton Theorem

The complexity class BPP describes a class of computational problems that are "practically feasible" because one may use pseudo-random resources to implement randomized algorithms. Therefore, it becomes extremely interesting to know whether NP can be solved by this model, i.e., whether NP $\subseteq$ BPP. If the answer is positive, then this would provide an exciting approach to solving NP-hard problems in practice. By the previous lecture, we know that BPP is a subclass of P/`poly`. Thus, NP $\subseteq$ BPP would imply NP $\subseteq$ P/`poly`. On the other hand, if it can be shown that NP $\subseteq$ P/`poly` does not hold true or is unlikely, then NP $\subseteq$ BPP also fails or becomes unlikely, thus excluding the possibility of solving NP-hard problems using randomized algorithms.

The purpose of this section is to present a famous result by Karp and Lipton [1] that shows that NP $\subseteq$ P/`poly` is unlikely.

Recall that the *polynomial-time hierarchy PH* is defined inductively as follows [2]:

$$\begin{aligned} & \Sigma_0^p = \Pi_0^p = \mathrm{P}, \\ \text{for all } k \geq 0, \qquad & \Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}, \quad \Pi_{k+1}^p = \text{co-}\Sigma_{k+1}^p, \ \text{ and} \\ & \mathrm{PH} = \bigcup_{k \geq 0} \Sigma_k^p. \end{aligned}$$

In particular, NP $= \Sigma_1^p$ and coNP $= \Pi_1^p$.

It is known [2] that for all $k \geq 1$, a language A is in Σ_k^p if and only if there is a polynomial-time computable Boolean function $F_A(x, y_1, y_2, \ldots, y_k)$ and a polynomial $p(n)$ such that:

$$A = \{x \mid \exists_p y_1 \forall_p y_2 \cdots Q_p y_k F_A(x, y_1, y_2, \ldots, y_k) = 1\},$$

where the quantifiers go alternatively between $\exists$ and $\forall$, starting from $\exists$. Thus, Q_p is either $\exists$ or $\forall$ depending on the parity of k. The subscript p on each quantifier Q_p means that the quantifier goes through all (binary) strings of length $p(|x|)$. Similarly, a language B is in Π_k^p if and only if there is a polynomial-time computable Boolean function $F_B(x, y_1, y_2, \ldots, y_k)$ and a polynomial $q(n)$ such that:

$$B = \{x \mid \forall_q y_1 \exists_q y_2 \cdots Q_q y_k F_B(x, y_1, y_2, \ldots, y_k) = 1\}.$$

It is commonly believed that the polynomial-time hierarchy PH does not collapse, that is, for all $k \geq 0$, we have $\Sigma_{k+1}^p \neq \Sigma_k^p$, or equivalently, PH $\neq \Sigma_k^p$ for any k.

Theorem 3.1 *If* NP $\subseteq$ P/`poly`*, then* $\Sigma_2^p = \Pi_2^p$.

PROOF. Let L be a language in Π_2^p. Thus,

$$L = \{x \mid \forall_p y_1 \exists_p y_2 F_L(x, y_1, y_2) = 1\},$$

where $F_L(x, y_1, y_2)$ is a polynomial-time computable function and p is a polynomial. Without loss of generality, we assume that if y_2 is a string of all 0's, then $F_L(x, y_1, y_2) = 0$ for all x and y_1 (otherwise, we can construct a function that satisfies this condition based on the given function F_L and keep the new function polynomial-time computable).

Consider the following language, where all b_h's are binary bits:

$$B = \{(x, y_1, b_1 b_2 \cdots b_{i-1}) \mid b_1 b_2 \cdots b_{i-1} 1 \text{ is a prefix of a string } y_2 \text{ such that } F_L(x, y_1, y_2) = 1\}$$

By the definition of the language L, if $F_L(x, y_1, y_2) = 1$, then $|y_1| = p(|x|)$, and $|y_2| = p(|x|)$. Therefore, in the above definition of B, we can assume that $|y_1| = p(|x|)$ and $1 \le i < p(|x|)$. The language B is clearly in NP: on an input $(x, y_1, b_1 b_2 \cdots b_{i-1})$, where $|y_1| = p(|x|)$ and $1 \le i < p(|x|)$, we can simply guess $p(n) - i$ binary bits $b'_{i+1}, \ldots, b'_{p(n)}$, let $y_2 = b_1 b_2 \cdots b_i 1 b'_{i+1}, \ldots, b'_{p(n)}$ then verify that $F_L(x, y_1, y_2) = 1$.

By our assumption NP $\subseteq$ P/`poly`, we have $B \in$ P/`poly`. By Theorem 2.1 in Lecture 2, B is accepted by a polynomial-size circuit family $\mathcal{F}_B = \{C_m \mid m \ge 1\}$.

Now fix an input length n, and let $m = n + p(n)$. We construct a circuit D_m that takes a binary string of length m as input and outputs a binary string of length $p(n)$. The purpose here is that for a pair (x, y_1), where $|x| = n$ and $|y_1| = p(n)$, if there is a y_2 such that $F_L(x, y_1, y_2) = 1$, then the circuit D_m on input (x, y_1) outputs such a y_2 with the largest lexicographic order.

The circuit D_m consists of the $p(n) + 1$ circuits C_m, C_{m+1}, ..., $C_{m+p(n)}$ in the circuit famity $\mathcal{F}_B$. For each $h \ge 0$, let b_h be the output of the circuit C_{m+h}. The output of D_m is $b_1 b_2 \cdots b_{p(n)}$. The input of the circuit C_m is (x, y_1), the same as the input of the circuit D_m. For each h, $0 \le h \le p(n) - 1$, the circuit C_{m+h+1} has $m + h + 1$ input bits: the first m input bits of C_{m+h+1} are from (x, y_1), the input of D_m, and the rest $h + 1$ input bits of C_{m+h+1} are $b_1 b_2 \cdots b_h b_0$. The circuit D_m is illustrated in Figure 1.

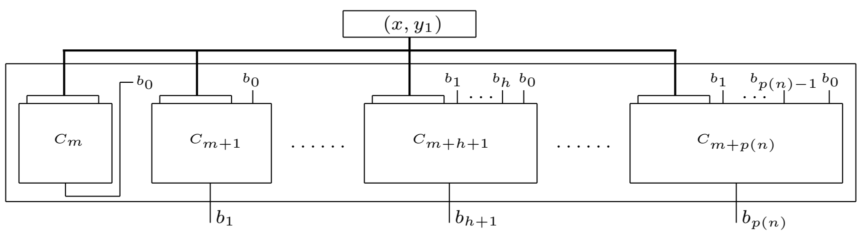

Figure 1: The circuit D_m

We prove that for the pair (x, y_1), where $|x| = n$ and $|y_1| = p(n)$, if there is a y_2 such that $F_L(x, y_1, y_2) = 1$, then the circuit D_m on input (x, y_1) outputs the y_2^0 of the largest lexicographic order such that $F_L(x, y_1, y_2^0) = 1$. Thus, assume that such a y_2^0 exists and $y_2^0 = b'_1 b'_2 \cdots b'_{p(n)}$. First note that under the assumption of the existence of y_2^0, the output b_0 of the circuit C_m is 1. Inductively, assume that the first h bits $b_1 b_2 \cdots b_h$ of $D_m(x, y_1)$ match the prefix $b'_1 b'_2 \cdots b'_h$ of y_2^0, where b_i is the output of the circuit C_{m+i} for $1 \le i \le h$. Consider the $(h + 1)$-st bit b_{h+1} of $D_m(x, y_1)$, which is the output of the circuit C_{m+h+1}. Note that the first m inputs of C_{m+h+1} are from (x, y_1) while the $(m + i)$-th input of C_{m+h+1} is the bit $b_i = b'_i$, for $1 \le i \le h$. Consider the $(h + 1)$-st bit b'_{h+1} of y_2^0. If $b'_{h+1} = 1$ then $b_1 b_2 \cdots b_h b_0 = b'_1 b'_2 \cdots b'_h b'_{h+1}$ is a prefix of y_2^0 (note $b_0 = 1$). Thus, $(x, y_1, b_1 b_2 \cdots b_h b_0)$ is in the language B so the output b_{h+1} of the circuit C_{m+h+1} on $(x, y_1, b_1 b_2 \cdots b_h b_0)$ should be 1, i.e., $b_{h+1} = b'_{h+1}$. If $b'_{h+1} = 0$ then $b_1 b_2 \cdots b_h b_0 = b'_1 b'_2 \cdots b'_h 1$ (again note $b_0 = 1$) cannot be a prefix of any y_2 that makes $F_L(x, y_1, y_2) = 1$ (otherwise, y_2^0 would not be such a string of the maximum lexicographic order). Thus, $(x, y_1, b_1 b_2 \cdots b_h b_0)$ is not in the language B so the output b_{h+1} of the circuit C_{m+h+1} on $(x, y_1, b_1 b_2 \cdots b_h b_0)$ is 0, which agains gives $b_{h+1} = b'_{h+1}$. This completes the inductive proof that $b_1 b_2 \cdots b_{p(n)} = b'_1 b'_2 \cdots b'_{p(n)} = y_2^0$. In conclusion, if there is a y_2 such that $F_L(x, y_1, y_2) = 1$, then on the input (x, y_1), the circuit D_m will output such a y_2^0 of the maximum lexicographic order.

On the other hand, if there is no y_2 that can make $F_L(x, y_1, y_2) = 1$, then for each $h \ge 0$, for any values of $b_1 b_2 \cdots b_h b_0$, $(x, y_1, b_1 b_2 \cdots b_h b_0)$ is not in the language B. Thus, the output b_{h+1} of the circuit C_{m+h+1} is 0. Therefore, in this case, the output $b_1 b_2 \cdots b_{p(n)}$ of the circuit D_m is $0^{p(n)}$. By our assumption, $F_L(x, y_1, 0^{p(n)}) = 0$. This proves the following critical fact:

> There is a y_2 of length $p(n)$ such that $F_L(x, y_1, y_2) = 1$ if and only if $F_L(x, y_1, D_m(x, y_1)) = 1$, where $D_m(x, y_1)$ stands for the output of the circuit D_m on input (x, y_1).

Since the circuits C_{m+h} for all h, $0 \le h \le p(n)$, have their size bounded by a polynomial of $n + p(n) + h \le$

$n+2p(n)$, which is also bounded by a polynomial of n, and since $p(n)$ is a polynomial of n, the circuit D_m has its size bounded by a polynomial of $n = |x|$.

Note that the circuit D_m works for all y_1 of length $p(n)$. Recall that the language L is given by

$$L = \{x \mid \forall_p y_1 \exists_p y_2 F_L(x, y_1, y_2) = 1\}.$$

Thus, the above observation shows that there is a circuit D_m such that a string x of length n is in L if and only if for all y_1, $|y_1| = p(n)$, $F_L(x, y_1, D_m(x, y_1)) = 1$.

Now we introduce a new Boolean function $F'_L(x, y_1, E_m)$, where $|x| = n$, $|y_1| = p(n)$, and E_m is a circuit of $m = n + p(n)$ inputs and $p(n)$ outputs, such that $F'_L(x, y_1, E_m) = 1$ if and only if $F_L(x, y_1, E_m(x, y_1)) = 1$. The function is certainly computable in polynomial time. Consider the following language (where p' is a polynomial that is the size of the circuit D_m given above):

$$L' = \{x \mid \exists_{p'} E_m \forall_p y_1 F'_L(x, y_1, E_m) = 1\}.$$

It is obvious that $L \subseteq L'$: for any $x \in L$, we have shown the existence of the circuit D_m such that $\forall_p y_1 F'_L(x, y_1, D_m) = 1$. On the other hand, for any $x' \in L'$, $|x'| = n$, by the definition, there is a circuit E_m of $n + p(n)$ inputs and $p(n)$ outputs such that for all y_1 of length $p(n)$ we have $F'_L(x', y_1, E_m) = 1$, which implies $F_L(x', y_1, E_m(x', y_1)) = 1$. Note that the output of $E_m(x', y_1)$ is a string y_2 of length $p(n)$. Thus, this implies that for each y_1, there is a $y_2 = E_m(x', y_1)$ such that $F_L(x', y_1, y_2) = 1$. Thus, x satisfies $\forall_p y_1 \exists_p y_2 F_L(x, y_1, y_2) = 1$ so is in L. This proves $L' \subseteq L$. Therefore, $L = L'$. Since L' by its definition is in Σ_2^p, we have proved that L is in Σ_2^p. Since L is an arbitrary language in Π_2^p, this proves $\Pi_2^p \subseteq \Sigma_2^p$.

The other direction can be easily derived. Let L' be a language in Σ_2^p, then its complement co-L' is in Π_2^p. By the above result, co-L' is also in Σ_2^p. Thus, the complement of co-L', which is L', is in Π_2^p. This gives $\Sigma_2^p \subseteq \Pi_2^p$, which completes the proof of $\Sigma_2^p = \Pi_2^p$. $\square$

Before we present the final Karp-Lipton Theorem, we notice the following well-known (and simple) fact on the polynomial-time hierarchy.

Lemma 3.2 *For any integer $k \geq 0$, if $\Sigma_k^p = \Sigma_{k+1}^p$, then* $\mathrm{PH} = \bigcup_{h \geq 0} \Sigma_h^p = \Sigma_k$.

PROOF. It suffices to prove that $\Sigma_k^p = \Sigma_{k+h}^p$ for all $h \geq 1$. We prove this by induction on h. The case $h = 1$ is given as the condition of the lemma. Now consider the general case $h \geq 2$. By the definition, $\Sigma_{k+h}^p = \mathrm{NP}^{\Sigma_{k+h-1}^p}$. By induction, we have $\Sigma_{k+h-1}^p = \Sigma_k^p$. Thus,

$$\Sigma_{k+h}^p = \mathrm{NP}^{\Sigma_{k+h-1}^p} = \mathrm{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p = \Sigma_k^p.$$

This completes the proof. $\square$

Now we are ready to prove the Karp-Liption Theorem.

Theorem 3.3 (Karp-Lipton) *If* $\mathrm{NP} \subseteq \mathrm{P}/\texttt{poly}$, *then* $\mathrm{PH} = \bigcup_{h \geq 0} \Sigma_h^p = \Sigma_2$.

PROOF. By Lemma 3.2, it suffices to prove that under the given condition, $\Sigma_2^p = \Sigma_3^p$.

Let L be a language in $\Sigma_3^p = \mathrm{NP}^{\Sigma_2^p}$. Thus, $L = M_1^B$, where M_1 is a nondeterministic polynomial-time oracle Turing machine that uses an oracle B in $\Sigma_2^p = \mathrm{NP}^{\Sigma_1^p}$ and accepts L. Thus, $B = M_2^C$, where M_2 is a nondeterministic polynomial-time oracle Turing machine that uses an oracle C in $\Sigma_1^p = \mathrm{NP}$ and accepts B. Under the condition $\mathrm{NP} \subseteq \mathrm{P}/\texttt{poly}$ given in the theorem, by Theorem 3.1, $\Sigma_2^p = \Pi_2^p$. Thus, the complement $\overline{B}$ of B is also in Σ_2^p. Thus, $\overline{B} = M_3^D$, where M_3 is a nondeterministic polynomial-time oracle Turing machine that uses an oracle D in NP and accepts $\overline{B}$. We can combine the two oracle sets C and D into a single set $H = 0C \cup 1D$, where $0C$ is the set obtained from C by inserting a 0 to the front of each element in C, and $1D$ is the set obtained from D by inserting a 1 to the front of each element in D. Since both C and D are in NP, the language $0C \cup 1D$ is also in NP.

Now we construct a new oracle Turing machine M that simulates the machine M_1 using the oracle H, as follows. M simulates M_1 step by step until M_1 makes a query y on its oracle B. Now the machine M nondeterministically decides to simulate either M_2 or M_3 on y. If M is simulating M_2 and M_2 accepts y (note that M_2 is a nondeterministic polynomial-time oracle machine that uses the oracle C but here in the simulation of M_2, M queries the elements in $0C$ in the oracle $H = 0C \cup 1D$ instead), then M knows that $y \in B$, so M can resume the simulation of M_1 with an answer "yes" to the query y. On the other hand, If M is simulating M_3 on oracle D and M_3 accepts y (in this simulation M queries the elements in $1D$ in the oracle $H = 0C \cup 1D$), then M knows that $y \in \overline{B}$, so $y \notin B$ and M can resume the simulation of M_1 with an answer "no" to the query y. Finally, if the simulation leads to a rejection of y (no matter in simulation of M_2 or M_3), then M simply rejects and stops.

Since all Turing machines M_1, M_2, and M_3 are nondeterministic and are running in polynomial-time, the Turing machine M is also nondeterministic and running in polynomial time. Since the oracle $H = 0C \cup 1D$ is in NP, the language accepted by M with oracle H is in $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$. We show that the machine M with oracle H accepts exactly the language L. For this, we only need to show that on each query y made by M_1 on the oracle B, the machine M is always able to get a correct answer to y by its simulation of M_2 or M_3. In fact, if $y \in B$, then the simulation of M_2 by M will have a computational path that accepts y, which will get a correct answer to the query y and M will continue the simulation of M_1 correctly. Similarly, for $y \notin B$ then a computational path in the simulation of M_3 by M get a correct answer to the query y and M will continue the simulation of M_1 correctly. Therefore, in all cases, there is a computational path of M that answers the query y and continues the simulation of M_1 correctly. On the other hand, if M is simulating a wrong machine (e.g., if $y \in B$ but M is simulating the machine M_3), or if M is simulating the right machine but on a wrong computational path (e.g., if $y \in B$ and M is simulating M_2 but is on a computational path that rejects y), then by our construction of the Turing machine M, that computational path of M is always stopped and has no impact on the entire computation of the Turing machine M on x). Since M_1 accepts the language L, the above discussion shows that the nondeterministic polynomial-time oracle Turing machine M with the oracle H in NP also accepts the language L. Therefore, $L \in \mathrm{NP}^{\mathrm{NP}} = \Sigma_2^p$. Since L is an arbitrary language in Σ_3^p, this shows $\Sigma_3^p \subseteq \Sigma_2^p$ so $\Sigma_3^p = \Sigma_2^p$. Now the theorem follows from Lemma 3.2. $\square$

References

[1] R. Karp and R. Lipton, Some connections between nonuniform and uniform complexity classes, in *Proc. 12th ACM Symposium on Theory of Computing*, pp. 302-309, (1980).

[2] L. J. Stockmeyer, The polynomial-time hierarchy, *Theoreticl Computer Science* 3-1, 1-22 (1976).