Complexity 2017.10.30
No Class on Nov. 2nd

Lecture 10. Some More Complexity Classes

10.1 Relativized complexity classes

The heart of polynomial-time reductions is to provide a way to design an algorithm for some problem A based on an assumed program for another problem B. Let us generalize this idea. Consider any problem X, and assume that X is solvable *magically* in $O(1)$ time. Then by using this assumed program for X, we design an algorithm for a given problem Y. If some program `IsY` solves Y in polynomial time by using the assumed program for X as a subroutine, then we say that Y is *polynomial-time solvable relative to* X. In other words, `IsY` is regarded as a program solving Y by asking queries to some *oracle* for X, an imaginary mechanism that can answer the question $y \in X$? instantly for any given y.

For any problem X, we define a class P^X to be the set of problems that are polynomial-time solvable relative to X. Classes NP^X, coNP^X, ... are defined similarly. These classes are called *relativized complexity classes*.

Relativized classes are usually used to discuss important open questions such as $\mathrm{P} \neq \mathrm{NP}$? in relativized worlds. But we can use this mechanism to define new complexity classes. For any complexity class $\mathcal{C}$, $\mathrm{P}^{\mathcal{C}}$ is the set of problems solvable in polynomial time relative to *some* problem in $\mathcal{C}$. For example, the class P^{NP} is one of the important complexity classes because this is the class corresponding to optimization problems related naturally to NP problems. (Here and in the following, we would sometimes use P (and in general, class P^X) to denote the class of polynomial-time computable functions (resp., relative to X).)

Here is an example of such optimization problems.

> Max. Clique Problem (MaxCLIQUEnum)
> **Instance**: An undirected graph G.
> **Question**: What is the size of the largest clique (i.e., complete subgraph) of G?

This is the problem that we saw at the first lecture. Remember at that time I explained that we could argue using the following decision problem because their computational difficulties are almost the same.

> Max. Clique Problem (decision version; abbreviated as CLIQUE)
> **Instance**: A pair $\langle G, k \rangle$ of an undirected graph G and an integer $k \geq 1$.
> **Question**: Does G have a clique of size $\geq k$?

However, precisely speaking, MaxCLIQUEnum is located in P^{NP} while CLIQUE is located in NP. And indeed we believe $\mathrm{NP} \neq \mathrm{P}^{\mathrm{NP}}$, and furthermore, MaxCLIQUEnum is a complete problem for P^{NP}. Thus, there is a difference!! But we have

$$\mathrm{P} = \mathrm{NP} \Leftrightarrow \mathrm{CLIQUE} \in \mathrm{P} \Leftrightarrow \mathrm{MaxCLIQUEnum} \in \mathrm{P},$$

and in this sense, these two problems can be regarded as problems with the almost same computational hardness w.r.t. P.

In general, the class P^{NP} is important for discussing the complexity of computing a "solution", that is, in our word, a witness for a positive instance. For example, the following problems.

3SAT Search Problem (search3SAT)
Instance: A 3CNF formula F.
Task: Find one of the satisfying assignments of F.
(Output 'no' if F has no sat. assignment.)

Hamiltonian Circuit Search Problem (searchHAM)
Instance: An undirected graph $G = (V, E)$, and $s, t \in V$.
Task: Find a Hamiltonian circuit.
(Output 'no' if F has no sat. assignment.)

10.2 The Polynomial-time Hierarchy and alternating complexity classes

Some of relativized complexity classes such as P^{NP}, $\mathrm{NP}^{\mathrm{NP}}$, etc., are also important for investigating some concrete problems. But classes like $\mathrm{NP}^{\mathrm{NP}}$ are not so easy to understand, at least not so intuitive. One way to get better understanding is to give a different formalization to these classes. For this we extend our definition of NP.

We simply generalize the *NP-condition* used to define the class $classNP$. For example, consider the following condition.

$$x \in L \;\Leftrightarrow\; \exists u : |u| \leq q_L(|x|),\, \forall v : |v| \leq q_L(|x|)\ [\ R_L(x, u, v)\ ].$$

A new complexity class Σ_2^{p} is the class of problems defined in this way with some polynomial q_L and polynomial-time computable predicate R_L. Similarly, classes Σ_3^{p}, Σ_4^{p}, ..., and classes Π_2^{p}, Π_3^{p}, ..., are defined using the following generalized NP-conditions.

$$\begin{aligned}
\Sigma_3^{\mathrm{p}}: &\quad x \in L \;\Leftrightarrow\; \exists_{q_L} u_1, \forall_{q_L} v_1, \exists_{q_L} u_2\ [\ R_L(x, u_1, v_1, u_2)\ ] \\
\Sigma_4^{\mathrm{p}}: &\quad x \in L \;\Leftrightarrow\; \exists_{q_L} u_1, \forall_{q_L} v_1, \exists_{q_L} u_2, \forall_{q_L} v_2\ [\ R_L(x, u_1, v_1, u_2, v_2)\ ] \\
&\quad \vdots \\
\Pi_2^{\mathrm{p}}: &\quad x \in L \;\Leftrightarrow\; \forall_{q_L} v_1, \exists_{q_L} u_1\ [\ R_L(x, v_1, u_1)\ ] \\
\Pi_3^{\mathrm{p}}: &\quad x \in L \;\Leftrightarrow\; \forall_{q_L} v_1, \exists_{q_L} u_1, \forall_{q_L} v_2\ [\ R_L(x, v_1, u_1, v_2)\ ] \\
&\quad \vdots
\end{aligned}$$

Below we will use the following notation. (Note that, by changing R_L appropriately, we may assume that witnesses are of the same polynomial length.)

$$\exists_{q_L} v_i \;\Leftarrow\; \exists v_i \in \{0,1\}^* : |v_i| = q_L(|x|), \quad \forall_{q_L} u_i \;\Leftarrow\; \forall u_i \in \{0,1\}^* : |u_i| = q_L(|x|).$$

Now using these classes, we can give the following characterization.

Theorem 10.1 (Wrathall '77 and Stockmeyer '77)

$$\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^{\mathrm{p}},\ \mathrm{coNP}^{\mathrm{NP}} = \Pi_2^{\mathrm{p}},\ \mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}} = \Sigma_3^{\mathrm{p}},\ \mathrm{coNP}^{\mathrm{NP}^{\mathrm{NP}}} = \Pi_3^{\mathrm{p}},\ \ldots$$

Proof by Idea. We show that $\mathrm{NP}^{\mathrm{NP}} \subseteq \Sigma_2^{\mathrm{p}}$.

Consider any problem L in $\mathrm{NP}^{\mathrm{NP}}$. Then there exist some relativized polynomial-time nondeterministic program $\mathtt{A}$ solving L by using some NP problem X as an oracle. Consider the computation of $\mathtt{A}$ on any input x of length ℓ, and let us formulate this computation by the Σ_2^{p} type formula. For the simplicity, we assume here that $\mathtt{A}$ asks exactly two queries on each computation path; moreover, assume that both queries are of length ℓ_1, where ℓ_1 is polynomially bounded by ℓ.

Each computation path is expressed by some binary string w of some fixed length p_1, where again p_1 is polynomially bounded by ℓ. If $\mathtt{A}$ were an ordinary nondeterministic program, its computation should be determined by such a binary string w. But since it makes two queries, the computation is not "easily" determined without knowing the answers of the oracle X to these queries. In fact, even the second query string may not be polynomial-time computable without knowing the answer to the first query. So, let us "guess" these two queries and the answers of X to them; note that each answer is expressed by one bit. That is, we express the computation of $\mathtt{A}(x)$ as follows.

$$
\begin{array}{l}
\mathtt{A} \text{ accepts } x \text{ (that is, } x \in L) \\
\quad \Leftrightarrow \quad \exists w \in \{0,1\}^{p_1},\, \exists y_1, y_2 \in \{0,1\}^{\ell_1},\, \exists b_1, b_2 \in \{0,1\} \\
\qquad \left[\begin{array}{l}
(1)\ [\ \text{the first query of } \mathtt{A} \text{ on the path } w \text{ is } y_1\ ] \\
\wedge\ (2)\ [\ \text{the second query of } \mathtt{A} \text{ on the path } w \text{ and with the answer } b_1 \text{ is } y_2\ ] \\
\wedge\ (3)\ [\ \mathtt{A} \text{ outputs } \mathtt{1} \text{ on } w \text{ and with the assumed answers } b_1 \text{ and } b_2\ ] \\
\wedge\ (4)\ [\ \text{the first query } y_1 \text{ gets 'yes' answer from } X \text{ iff } b_1 = 1\ ] \\
\wedge\ (5)\ [\ \text{the second query } y_2 \text{ gets 'yes' answer from } X \text{ iff } b_2 = 1\ ]
\end{array} \right]
\end{array}
$$

It is easy to see that (1) $\sim$ (3) can be checked *deterministically* in polynomial time. On the other hand, this may not be the case for (4) and (5) because $X \in \mathrm{NP}$. But, for example, (4) can be expressed as follows with some q_X and R_X.

$$
\begin{array}{rcl}
(4) & \Leftrightarrow & [\, b_1 = 1 \wedge [\, y_1 \in X\,]\,] \ \vee\ [\, b_1 = 0 \wedge [\, y_1 \notin X\,]\,] \\
& \Leftrightarrow & [\, b_1 = 1 \wedge \exists_{q_X(\ell_1)} u_1\, [\, R_X(y_1, u_1)\,]\,] \ \vee\ [\, b_1 = 0 \wedge \forall_{q_X(\ell_1)} v_1\, [\, \neg R_X(y_1, v_1)\,]\,] \\
& \Leftrightarrow & \exists_{q_X(\ell_1)} u_1,\, \forall_{q_X(\ell_1)} v_1\ [\ \underbrace{[\, b_1 = 1 \wedge R_X(y_1, u_1)\,] \ \vee\ [\, b_1 = 0 \wedge \neg R_X(y_1, v_1)\,]}_{(4')}\ ]
\end{array}
$$

Hence, as a whole, we can restate "$\mathtt{A}$ accepts x" by the following Σ_2^{p}-type formula.

$$\exists_{p_1} w, \exists_{\ell_1} y_1, y_2, \exists_1 b_1, b_2, \exists_{q_X(\ell_1)} u_1, u_2, \forall_{q_X(\ell_1)} v_1, v_2\ [\ (1) \sim (3) \wedge (4') \wedge (5')\ ].$$

□

We may regard $\mathrm{NP} = \Sigma_1^{\mathrm{p}}$ and $\mathrm{coNP} = \Pi_1^{\mathrm{p}}$. Also define $\Sigma_0^{\mathrm{p}} = \Pi_0^{\mathrm{p}} = \mathrm{P}$. For relativized classes like P^{NP}, we introduce the class $\Delta_k^{\mathrm{p}} = \mathrm{P}^{\Sigma_{k-1}^{\mathrm{p}}}$. Then the following relationship holds from the definition or as a corollary of Theorem 9.1.

Corollary 10.2

$$\mathrm{P} = \Sigma_0^{\mathrm{p}} = \Pi_0^{\mathrm{p}} \ \subseteq\ \mathrm{NP} = \Sigma_1^{\mathrm{p}}, \mathrm{coNP} = \Pi_1^{\mathrm{p}} \ \subseteq\ \Sigma_2^{\mathrm{p}}, \Pi_2^{\mathrm{p}} \ \subseteq\ \Sigma_3^{\mathrm{p}}, \Pi_3^{\mathrm{p}} \ \subseteq\ \cdots$$

Corollary 10.3 $\Sigma_{k-1}^{\mathrm{p}} \cup \Pi_{k-1}^{\mathrm{p}} \subseteq \Delta_k^{\mathrm{p}} \subseteq \Sigma_k^{\mathrm{p}} \cap \Pi_k^{\mathrm{p}}$.

Polynomial-time hierarchy PH is defined to be the union of these Σ_k^{p} classes.

Notice that the order of quantifiers cannot be changed in general (unless they are of the same type). Thus, from our Σ_2^{p} formulation of $\mathrm{NP}^{\mathrm{NP}}$, it is clear that $\mathrm{NP} \neq \mathrm{NP}^{\mathrm{NP}}$ unless $\mathrm{NP} = \Pi_1^{\mathrm{p}}$ (= coNP). In fact, it is easy to see that $\mathrm{NP} = \mathrm{NP}^{\mathrm{NP}}$ (and thus, the whole PH collapses NP) if and only if NP = coNP.

Corollary 10.4 $\mathrm{NP} = \mathrm{coNP} \Leftrightarrow \mathrm{NP} = \mathrm{PH}$. In general, $\Sigma_k^{\mathrm{p}} = \Pi_k^{\mathrm{p}} \Leftrightarrow \Sigma_k^{\mathrm{p}} = \mathrm{PH}$.

The alternation of quantifiers can be viewed as a play of some two player game. For example, consider the following *quantified Boolean formula satisfiability problem* Q.

Quantified Boolean Formula satisfiability Problem (QBF-SAT)
Instance: A formula of the form
$$Q \;=\; \exists_q u_1,\, \forall_q v_1,\, \exists_q v_2,\, \forall_q u_2 \;\; [\; \Phi(x, u_1, v_1, u_2, v_2) \;] \,.$$
Question: Is this Q satisfiable?

The existential quantifier is for (the choice of) you, and the universal quantifier is for (the choice of) the opponent. The goal of the game (from your point of view) is to make Φ true; on the other hand, your opponent's goal is to make Φ false. The number of alternations can be regarded as the number of turns (or the depth) of the game. (Here for the simplicity, we assume that every formula starts with an existential quantifier and ends with an universal quantifier.)

Let us call this game an *alternating satisfying game* (over a *polynomial-time* Boolean predicate). Note that each game is defined by a predicate Φ that, we assume, polynomial-time computable, and each play of the game is defined by a given input x. We say that an alternating satisfying game *solves* a problem L if $x \in L$ if and only if the $\exists$-player wins at the play for x. We can further generalize this two player alternating satisfying game so that the depth of each play is not constant but $d(|x|)$ on each input x for some given function. With this computation model, we can introduce the following new complexity class.

$$\mathrm{ADepth}(d) \;=\; \left\{ L \;:\; \begin{array}{l} L \text{ is solvable by some alternating satisfying game} \\ \text{whose game depth is } d(|x|) \text{ on each input } x \end{array} \right\}.$$

Then we have $\mathrm{PH} = \mathrm{ADepth}(O(1))$. What is interesting here is that the class PSPACE is characterized in the following way.

Theorem 10.5 $\mathrm{PSPACE} = \bigcup_{p:\mathrm{poly}} \mathrm{ADepth}(p)$.

Proof. Consider any problem L in PSPACE, and let $\mathtt{A}$ be a deterministic program solving L, whose space complexity is bounded by some polynomial p.

Consider the execution of $\mathtt{A}$ on any input of length ℓ. Since strings kept in $\mathtt{A}$'s each register is at most of $p(\ell)$ bits, the state of the program at each step can be expressed by a binary string of length $q = cp(\ell)$ for some constant $c > 0$. In fact, with these states and direct edges connecting them, we can express the computation of $\mathtt{A}$ on *all* inputs of length ℓ as a graph $G_{\mathtt{A},\ell}$.

More specifically, the graph $G_{\mathtt{A},\ell}$ consists of vertices corresponding states, each of which is labeled by binary string in $\{0,1\}^q$, and the graph has an edge from one vertex S_1 to

another S_2 if and only if the execution of $\mathtt{A}$ moves from S_1 to S_2 in one step. Then for any input $x \in \{0,1\}^\ell$, we have $x \in L$ if and only if there is a path from the *initial vertex* for x, the vertex corresponding to the initial state of $\mathtt{A}(x)$, to the *accepting vertex*, a vertex corresponding to the halting state yielding 1 (i.e., 'yes'). Notice here that the length of such an *accepting path* is at most 2^q; in fact, we may assume that the length is exactly $2^{d(\ell)}$ for some polynomial d.

Now with this graph, we define an alternating satisfying game for simulating $\mathtt{A}$ on x for any given $x \in \{0,1\}^\ell$. Intuitively, you want to convince the opponent that there is a path from the initial vertex S_0 for x to the accepting vertex $S_{\rm acc}$. For this, you point out the vertex S_1 that is located exactly at the middle of the path. Then the opponent asks you to show either (i) there is a path from S_0 to S_1, or (ii) there is a path from S_1 to $S_{\rm acc}$. In other words, the opponent's move is either 0 indicating (i) or 1 indicating (ii). Then depending on this move, you show for S_2 either (0) the vertex located at the middle of S_0 and S_1, or (1) the vertex located at the middle of S_1 and $S_{\rm acc}$. The play proceeds in this way for $d(\ell) - 1$ pairs of moves. Then after these moves (if your choices have been correct), there must be an edge between the current vertices S_i and S_j, i.e., $\mathtt{A}$ moves from S_i to S_j in one step. (The indices i and j are determined easily from the sequence of the opponent moves.)

Since you cannot predict this opponent's move in advance, you have to give the correct middle vertex for S_1, and so on; otherwise, the opponent certainly finds the problematic part. In particular, if there is no accepting path for a given x, then you cannot win the game no matter how you choose moves. Therefore, we have the following relation, which proves that $L \in \mathrm{ADepth}(d')$ for $d'(n) = 2d(n) - 2$.

$$x \in L \;\Leftrightarrow\; \exists S_1, \forall v_1 \in \{0,1\}, \exists S_2, \forall v_2 \in \{0,1\}, \ldots \exists S_{d(\ell)-1}, \forall v_{d(\ell)-1} \in \{0,1\} \;[\; S_i \Rightarrow_{\mathtt{A}} S_j \;].$$

□

One remark here. The class $\mathrm{ADepth}(d)$ is introduced for the explanation in this lecture, and it is not commonly used. Standard complexity classes related to this $\mathrm{ADepth}(d)$ is alternating complexity classes that are defined by using *alternating computation*, which is a generalization of nondeterministic computation.

Homework exercise from Lecture 10

Homework rule: Choose one of the following problems, and hand your answer in at the next class (i.e., Nov. 6th, i.e., the next week). (If you cannot attend the next class, please submit your answer via email before the class.) Since we have one week this time, you can solve more than one problem (if you like) to get more points.

* For writing an answer, you may use Japanese.

Basic problems

1. Prove that if NP = coNP, then NP = PH.

An advanced problem

1. Suppose that NP = coNP and prove formally* that the following Lexfirst3SAT problem is in NP. (∗ Need to prove based on the definition of the class NP.)

   Lexfirst3SAT
   Instance: 3CNF formula F over n variables, and an integer i, $1 \leq i \leq n$.
   Question: What is the ith bit of the lexicographically the first satisfying assignment of F? (Yes/No $\Leftrightarrow$ 1/0)

2. Prove that the above Lexfirst3SAT problem is complete for P^{NP}. That is, every problem in P^{NP} is $\leq_{\mathrm{m}}^{\mathrm{P}}$-reducible to Lexfirst3SAT.