In this lecture we talk about the conception of advice which is an extra input to a Turing machine that is allowed to only depend on the length $n$ of the input, but not the input itself. There are a variety of classes such as $P/poly$, $L/poly$ and $NL/poly$ with some interesting conclusions. And then we move on the Karp-Lipton theorem which involves $NP$ and $P/poly$. Finally, we introduce the notion of selector function of an language \cite{1}.

1 Advice and Nonuniform Computation Models

Definition 1 (Advice). Given complexity class $C$ and advice size bound $a: \mathbb{N} \rightarrow \mathbb{N}$, define

$$C/a(n) = \{ L \mid (\exists L' \in C)(\exists \alpha_0, \alpha_1, \alpha_2, \ldots \in \Sigma^*) \text{ such that } x \in L \iff (x, \alpha_{|x|}) \in L' \} ,$$

where $|\alpha_n| \leq a(n)$.

Based on the length of the bits $n$, the advice function give an output of each $n$. The bound or the length of the advice string is only depended on the input size in the class $C$. $P/poly$ is the class of all languages that can be computed in polynomial time with polynomial-length advice. We may view that a Turing machine is added a single advice polynomial length string $a(n)$ with all inputs of length $n$, or a infinite series of Turing machine and the $n$th Turing machine computes of input of length $n$.

$P/poly$ have the relations with boolean circuits which can used as an general and simplified abstract model of the basic part in modern computers. The boolean circuit is such kind of nonuniform model that allows a different algorithm to be used for each input size. While there is always the same Turing machine solving problems with infinite input sizes in the standard Turing machine model. Additionally, proving the lower bounds of Boolean circuits seems to be easier than proving the lower bounds of Turing machine. $P/poly$ is the class of all languages that can be computed in polynomial time with polynomial-length advice.

Theorem 1. $L \in P/poly$ if and only if there exists a family of circuits $\{C_n\}$, where circuit $C_n$ has $n$ inputs and one output, and there exists a polynomial $p(\cdot)$ such that for all $n$, $|C_n| \leq p(n)$ and $C_n$ decides language $L$ on $x \in \{0,1\}^n$. The corresponding language is

$$P/poly = \{ L \mid \exists c \forall n C_L(n) < n^c \}$$

where $C_L(n)$ is the size of a smallest Boolean circuit that decides $L$ at length $n$. 

1
Proof. \((\Rightarrow)\): Based on the definition of advice, there exists a Turing machine \(M\) and a sequence of advice \(\{\alpha_n\}\) deciding the language \(L\). \(M(\alpha_n, \cdot)\) is a polynomial time-bounded deterministic Turing machine working on \(n\)-length inputs. We consider a non-deterministic machine that accepts \(x\) inputs within \(t\) steps, or a deterministic machine that accepts \((x, y)\) within \(t\) steps and the length of \(y\) is polynomial in \(|x|\). That is the instance of a Bounded Halting problem \((\langle M \rangle, x, 1^t)\). The polynomial transformation of this instance is comprised of constructing a circuit \(C\) which on input \(y\) outputs \(M(x, y)\) because a circuit can be constructed which on input \(x\) outputs \(M(\alpha_n, x)\) to imitate the \(M(\alpha_n, \cdot)\). In this way, we build a sequence \(\{C_n\}\) of circuits, where for each \(n\), \(C_n\) is an encoding of \(M(\alpha_n, \cdot)\).

\((\Leftarrow)\): There exists a family \(\{C_n\}\) of circuits deciding language \(L\) and the size of each \(C_n\) is polynomial in \(n\). We can use a standard encoding method to describe \(C_n\). Given a universal Turing machine \(U\) such that for all \(n\), and all \(x\) of length \(n\). We use this universal Turing machine to simulate \(C_n(x)\) using the \(C_n(x)\) and \(x\) as input. Actually, the set of the description of the \(C_n\) can be seen as the advice strings \(\alpha_n\) for every length \(n\). So \(U(\alpha_n, x)\) simulates \(C_n(x)\). Bases on the definition of advice, \(L \in P/poly\).

Similarly, when we considered the complexity class \(L\), the advice of \(L\) is \(L/poly\).

**Theorem 2.** \(A \in L/poly\) if and only if there exists a family of branching programs \(\{BP_n\}\) where for each \(n\), \(BP_n : \{0, 1\}^n \rightarrow \{0, 1\}\), and there exists a polynomial \(p(\cdot)\) such that \(|BP_n| \leq p(n)\) and \(BP_n\) decides language \(A\) for all \(x \in \{0, 1\}^n\). The corresponding language is

\[
L/poly = \{A \mid \exists c \forall n \ BP_A(n) < n^c\}
\]

where \(BP_A(n)\) is the size of the smallest branching program that decides a language \(A\) at length \(n\).

Proof. \((\Rightarrow)\): The proof is similar with the above. There exists a advice function \(\alpha\) mapping an integer \(n\) to a string of length polynomial in \(n\), and a Turing machine \(M\) with tapes of size logarithmic in the input size. Because \(L\) has a input \(x\) with length \(n\), \(M\) decides \(L\) and accepts the input \((x, \alpha)\). The configuration of each step of \(M\) can be seen as the node in the branching program. At each node, the program look at the bit of input (0 or 1) corresponding to this node’s label. It move along the outgoing 0 edge to the next node, likewise take the 1 edge if the bit is a 1. The configuration state of \(M\) change step by step also in this way. Because we just need to keep track of the current node, running an advice-encoded BP only requires logarithmic space.

\((\Leftarrow)\): If a branching program of polynomial size exists, it can be specified by the advice function and simulated by the Turing machine. The size of the branching program is reduced to \(O(n)\), meaning that the advice function is polynomial bound.

Similarly, we can get a the advice of \(NL\).

**Theorem 3.** \(L \in NL/poly\) if and only if there exists a family of nondeterministic branching program \(\{NBP_n\}\) where for each \(n\), \(NBP_n : \{0, 1\}^n \rightarrow \{0, 1\}\), and there exists a polynomial \(p(\cdot)\) such that \(|NBP_n| \leq p(n)\) and \(NBP_n\) decides language \(A\) for all \(x \in \{0, 1\}^n\). The corresponding language is

\[
NL/poly = \{A \mid \exists c \forall n \ NBP_A(n) < n^c\}
\]

where \(NBP_A(n)\) is the size of the smallest branching program that decides a language \(A\) at length \(n\).
2 Karp-Lipton Theorem

One of the motivation to define \( P/poly \) is to separate \( P \) from \( NP \). If we can shows there is a language which is in \( NP \) but not in \( P/poly \) and not in \( P \), we would like to show that \( P \neq NP \). Karp-Lipton theorem shows that if \( NP \subseteq P/poly \), the Polynomial-time hierarchy will collapse to \( \Sigma_2^P \cap \Pi_2^P \).

**Theorem 4.** If \( NP \subseteq P/poly \), then \( \text{PH} = \Sigma_2^P \).

**Proof.** If \( NP \subseteq P \), then \( SAT \subseteq P \) which means that there is a deterministic polynomial-time Turing machine \( M \), a polynomial function \( p(\cdot) \) and advice strings \( \{\alpha_n\}_{\forall n \in \mathbb{N}, |\alpha_n| \leq p(n)} \), and for every boolean formula \( F \), \( M \) accepts \( (F, \alpha_F) \) if and only if \( F \) is satisfiable. Thus, the assumption means that \( SAT \) has polynomial circuits. By proving \( \Pi_2^P \subseteq \Sigma_2^P \) we can get \( \text{PH} = \Pi_2^P = \Sigma_2^P \).

Consider a language \( L \in \Pi_2^P \), there is a polynomial time predicate \( V \) so that

\[
x \in L \iff (\forall y_1 \in \Sigma^{|x|^e}) \exists y_2 \in \Sigma^{|x|^e} [(x, y_1, y_2) \in V]
\]

where \(|y_1|\) and \(|y_2|\) are polynomially bounded in \(|x|\). Note that \( "(\forall y_2 \in \Sigma^{|x|^e}) [(x, y_1, y_2) \in V]" \) is an NP question. Thus, it can be reduced to an instance of SAT. The above formula can be transformed to

\[
x \in L \iff (\forall y_1 \in \Sigma^{|x|^e}) [\varphi(x, y_1) \in SAT]
\]

where \( \varphi(., .) \) is a polynomial time computable function bounded in \(|x|\). Based on the assumption, there exists a polynomial space circuit which can work correctly for SAT for all formulas of length at most \( p(n) \). Here \( p(\cdot) \) is some polynomial and \( n = |x| \). We can firstly get a circuit \( C \) using \( \Sigma_2^P \) computation. This circuit may output faults, which are rejecting a \( \varphi \) in SAT and/or accepting a \( \varphi \) and SAT. We can convert \( C \) into a \( C' \) which may only output one kind of fault which is rejecting a \( \varphi \) in SAT. We want to check for all phase whether \( C' \) will accept the input \( \varphi(x, y_1) \), that is

\[
(\forall y_1 \in \Sigma^{|x|^e}) [C'(\varphi(x, y_1)) = 1].
\]

If \( x \in L \), there exist a polynomial circuit \( C^* \) and \( C' \) will accept all of the for all polynomial bounded \( y_1 \). Our simulation of the \( C^* \) will always guess this circuit and have an accepting path. If \( x \notin L \). Then, there is some \( y_1 \) for that \( \varphi(x, y_1) \) is not satisfied. For all the \( C \) in the process of \( \Sigma_2^P \) computation, \( C' \) will reject \( \varphi(x, y_1) \). Now the important point is how to get such \( C' \), we can do it as follows.

Assume we have guessed the circuit \( C \) in polynomial time. To get \( C' \), we ask oracle query to circuit \( C \). By doing this, we want to decide if a boolean formula is satisfiable and generate a satisfying assignment if it is so. Because \( SAT \in P/poly \) implies that the SAT searching problem is also in \( P/poly \), the SAT searching problem has a polynomial-sized circuit that can guessed in a polynomial time. So we do the \( \Sigma_2^P \) computation and guess the \( C' \) firstly. And then we are in a universal state and start to generate a \( \phi \) which is polynomial bounded in \(|X|\) for all \( y_1 < \{0, 1\}^{|x|^e} \). Then we evaluate \( C'(\phi(x, y_1)) \). If \( C' \) decides that \( \phi(x, y_1) \) is not satisfiable, we rejects. If \( C' \) decides that \( \phi(x, y_1) \) is satisfiable it goes ahead and computes a satisfying assignment \( \phi \) for \( \varphi(x, y_1) \). To generate the \( \phi \), we use the self-reduction of SAT.
We input $\varphi(x_1, x_2, \ldots, x_m)$ to $C$. For every input, if $C$ rejects, $C'$ rejects. If $C$ accepts, $C'$ manages to find a satisfying truth assignment $\phi$ to $\varphi$. Firstly, we set all the occurrences of $x_1$ in $\varphi$ to be TRUE to generate a new $\varphi_1$. Secondly, we ask $C$ if $\varphi_1$ is satisfiable. If it is satisfiable, then $\phi(x_1) = 1$, otherwise $\phi(x_1) = 0$. Then we repeat the above process. In another word, we set all the occurrences of $x_2$ in $\varphi_1$ to be TRUE to generate a new $\varphi_2$ and ask $C$ if $\varphi_2$ is satisfiable. If it is satisfiable, then $\phi(x_2) = 1$, otherwise $\phi(x_2) = 0$, and so on. Finally, we get a truth assignment $\phi$. Then we check whether $\phi$ really satisfies $\varphi$, if so $C'$ accepts, else $C'$ rejects. In this way $C'$ only accepts when is has verified a satisfying truth assignment. $C'$ makes errors only when it reject a satisfiable formula.

Exercises:

- $\text{NP} \subseteq P/O(\lg n) \Rightarrow \text{NP} = \text{P}$;
- $\text{NP} \subseteq P/O(\lg n) \Rightarrow \text{NL} = \text{L}$.

3 Selector of an Language

For a given language $L$, instead of solving the decision problem of $L$ for an arbitrary input $x$, we may be more interested: which one of two given input strings $x$ and $y$ is more likely to be in $L$? The notion of P-selective sets is presented to study the polynomial-time reproducibilities on NP. It plays a great role in researching several important structural concepts such as self-reducibility and reducing search to decision and function complexity classes [2].

**Definition 2.** Let $\Sigma$ be a finite alphabet. A language $L \subseteq \Sigma^*$ is said to be P-selective exactly if there is a function $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ which is computable in polynomial time such that

- $f(x, y) \in \{x, y\};$
- if $\{x, y\} \cap L \neq \emptyset$, then $f(x, y) \in L$.

The function $f$ is said to be a P-selector function for language $L$.

**Theorem 5.** SAT is P-selective if and only if $P = \text{NP}$.

**Proof.** Assume $f$ is a selector for SAT. Note that for any two formulas of propositional logic $\phi$ and $\varphi$, we have $f(\phi, \varphi) \leftrightarrow (\phi \lor \varphi) \in \text{SAT}$.

Let $\phi$ be in conjunctive normal form. Suppose $\phi = C_1 \land C_2 \ldots \land C_n (n \geq 1)$, and each $C_i = L_{i_1} \lor L_{i_2} \lor \ldots \lor L_{i_m}$, there each $L_{i_j}$ is a literal. To determine in polynomial time whether $\phi \in \text{SAT}$, let $\phi$ be as follows:

$$\phi = C_1 \land \ldots \land C_n = (L_{i_1} \lor L_{i_2} \lor \ldots \lor L_{i_{m_1}}) \land \ldots \land (C_2 \lor \ldots \lor C_n).$$

We distribute the first set of disjuncts. Let $D_j = (L_{1_j} \land C_2 \lor \ldots \lor C_n), j \leq m_1$. Compute $f(D_1, D_2 \lor D_3 \lor \ldots \lor D_{m_1})$. If the answer is $D_1$ then $\phi$ is satisfiable only if $D_1$ is satisfiable, so proceed with $D_1$. Otherwise, compute $f(D_2, D_3 \lor \ldots \lor D_{m_1})$. Continue this process until some $D_j$ is obtained. So there is

$$D_j = L_{1_j} \land C_2 \land \ldots \land C_n = L_{1_j} \land (L_{2_{1_j}} \lor L_{2_{2_j}} \lor \ldots \lor L_{2_{m_2}}) \land \ldots \land (C_2 \land \ldots \land C_n).$$
Repeat the above process $C_2$ putting place of $C_1$ and repeat the above process. Eventually we can obtain a conjunction of literals (one literal from each $C_i$). $\phi$ is satisfiable if and only if this conjunction of literals is satisfiable. Verify whether or not this is so. The results is a polynomial time algorithm.

4 Next Time

In the next lecture, we are going to talk about the kernelization lower bound. The statement of the kernelization lower bound is: Vertex Cover does not have kernels consisting of $O(k^{2-\epsilon})$ edges for any positive constant $\epsilon$ unless coNP is in NP/poly. Recall that Vertex Cover is the problem of deciding whether, given a graph $G$ and an integer $k$, $G$ has a vertex cover of size at most $k$. A kernel is a mapping reduction from a parameterized problem to itself such that the size of the reduced instance is bounded by a function of the parameter only. Kernelization gives us quantitative insights in what can be achieved by polynomial time preprocessing.

5 Acknowledgements

In writing the nodes for this lecture, we perused the notes by David Guild for Lecture 9 from the Spring 2011 offering of CS 710.

References
