Lecture 20

NP, polynomial-time mapping reductions, and NP-completeness

In the previous lecture we discussed deterministic time complexity, along with the time-hierarchy theorem, and introduced two complexity classes: P and EXP. In this lecture we will introduce another complexity class, called NP, and study its relationship to P and EXP. In addition, we will define a polynomial-time variant of mapping reductions along with the notion of completeness for the class NP.

Remark 20.1. The concept of NP-completeness is certainly among the most important contributions theoretical computer science has made to science in general; NP-complete problems, so recognized, are ubiquitous throughout the mathematical sciences. It is therefore fitting that this concept should be mentioned in a course titled *Introduction to the Theory of Computing*.

At the University of Waterloo, however, the complexity class NP and the notion of NP-completeness are covered in a different course, *CS 341 Algorithms*. For this reason, in the present course we do not place the focus on these notions that they deserve—here we are essentially just treating the class NP as an important example of a complexity class. In particular, we will not cover techniques for proving specific languages are NP-complete, but those new to this topic may rest assured that thousands of interesting examples, including ones of great practical importance, are known.

20.1 The complexity class NP

There are two equivalent ways to view the complexity class NP. The first way is where NP gets its name, as the class of languages decidable in *nondeterministic polynomial time*. The second way, which is more intuitive and more easily applied,

is as the class of languages that are *verifiable in polynomial time*, given a polynomial-length *certificate* for the membership of any string in the language in question.

NP as polynomial-time nondeterministic computations

As suggested above, we may define the class NP as its name suggests, as *nondeterministic polynomial time*.

To begin, let us first be precise about what is meant by the running time of a nondeterministic Turing machine.

Definition 20.2. Let N be an NTM having input alphabet Σ. For each input string $w \in \Sigma^*$, let $T(w)$ be the *maximum* number of steps (possibly infinite) for which N runs on input w, over all possible nondeterministic computation paths. The *running time* of N is the function $t : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ defined as

$$t(n) = \max\{T(w) \,:\, w \in \Sigma^*,\ |w| = n\} \tag{20.1}$$

for every $n \in \mathbb{N}$. In words, $t(n)$ is the maximum number of steps required for N to halt, over all input strings of length n and over all nondeterministic choices of N.

As in the previous lecture, we shall restrict our attention to NTMs having finite running times for all input lengths.

Let us recall explicitly from Lecture 18 how decidability is is characterized by nondeterministic Turing machines: an NTM N decides a language A if $A = \mathrm{L}(N)$ and N has a finite computation tree for every input string. Thus, if N is an NTM having running time t and w is an input string to N, the computation tree of N on input w always has depth bounded by $t(|w|)$. It is an *accepting* computation if there exists at least one accepting configuration in the computation tree, and otherwise, if there are no accepting configurations at all in the tree, it is a *rejecting* computation.

We may now define $\mathrm{NTIME}(f)$, for every function $f : \mathbb{N} \to \mathbb{N}$, to be the complexity class of all languages that are decided by a nondeterministic Turing machine running in time $O(f(n))$. Having defined $\mathrm{NTIME}(f)$ in this way, we define NP as we did for P:

$$\mathrm{NP} = \bigcup_{k \geq 1} \mathrm{NTIME}(n^k). \tag{20.2}$$

NP as certificate verification

The second way to view NP is as the class of languages that are verifiable in polynomial time, given polynomial-length certificates for membership. We shall state

a theorem that equates this notion to membership in the complexity class NP, as defined by (20.2).

Before we state the theorem, however, let us introduce some simplifying terminology. Hereafter, a function of the form $p : \mathbb{N} \to \mathbb{N}$ is a *polynomially bounded function* if it is time constructible and satisfies $p(n) = O(n^k)$ for some positive integer k.

Theorem 20.3. *Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language. The language A is contained in* NP *if and only if there exists a polynomially bounded function p and a language $B \in$ P such that*

$$A = \Big\{ x \in \Sigma^* \,:\, \exists y \in \{0,1\}^* \text{ such that } |y| \leq p(|x|) \text{ and } \langle x, y \rangle \in B \Big\}. \tag{20.3}$$

An interpretation of this theorem is that the string y plays the role of a *certificate* or *proof* that a string x is contained A, while the language B represents an efficient *verification procedure* that checks the validity of this proof of membership for x. The terms *witness* is another alternative name to certificate.

The basic idea behind the proof of two implications needed to imply the theorem, which we shall not cover in detail, are as follows.

1. Given a language A characterized by the equation (20.3) in the theorem statement, one may conclude that $A \in$ NP by defining a polynomial-time NTM N so that it first nondeterministically guesses a binary string y having length at most $p(|x|)$ and then decides membership of the string $\langle x, y \rangle$ in B.
2. If $A \in$ NP, so that we have a polynomial-time NTM N that decides A, we may encode each nondeterministic computation path of N on any input x by some binary string y having length at most $p(|x|)$, for some polynomially bounded function p. We then define B to be the language of strings $\langle x, y \rangle$ for which y encodes an accepting computation path of N on input x, which can be decided in polynomial time.

Relationships among P, NP, and EXP

Let us now observe the following inclusions:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}. \tag{20.4}$$

The first of these inclusions, P $\subseteq$ NP, is straightforward. Every DTM is equivalent to an NTM that never has multiple nondeterministic options, so

$$\mathrm{DTIME}(f) \subseteq \mathrm{NTIME}(f) \tag{20.5}$$

The DTM M operates as follows on input $x \in \Sigma^*$:

1. Set $y \leftarrow \varepsilon$.
2. If $\langle x, y \rangle \in B$, then accept.
3. Increment y with respect to the lexicographic ordering of the binary alphabet.
4. If $|y| > p(|x|)$ then reject, else goto step 2.

Figure 20.1: A DTM M that decides a given NP-language in exponential time.

for all functions $f : \mathbb{N} \to \mathbb{N}$. This implies that

$$\text{DTIME}(n^k) \subseteq \text{NTIME}(n^k) \tag{20.6}$$

for all $k \geq 1$, and therefore $\text{P} \subseteq \text{NP}$.

Alternatively, with respect to the characterization of NP given by Theorem 20.3, suppose $A \subseteq \Sigma^*$ is a language over an alphabet Σ, and assume $A \in \text{P}$. We may then define a language B as follows:

$$B = \big\{ \langle x, \varepsilon \rangle \,:\, x \in A \big\}. \tag{20.7}$$

It is evident that $B \in \text{P}$; as A in polynomial time, we can easily decide B in polynomial time as well. For any choice whatsoever of a polynomially bounded function p it is the case that (20.3) holds, and therefore $A \in \text{NP}$.

Now let us observe that $\text{NP} \subseteq \text{EXP}$. Suppose $A \subseteq \Sigma^*$ is language over an alphabet Σ, and assume $A \in \text{NP}$. By Theorem 20.3, there exists a polynomially bounded function p and a language $B \in \text{P}$ such that (20.3) holds. Define a DTM M as described in Figure 20.1. It is evident that M decides A, as it simply searches over the set of all binary strings y with $|y| \leq p(|x|)$ to find if there exists one such that $\langle x, y \rangle \in B$.

It remains to consider the running time of M. Let us first consider step 2, in which M tests whether $\langle x, y \rangle \in B$ for an input string $x \in \Sigma^*$ and a binary string y satisfying $|y| \leq p(|x|)$. This test takes a number of steps that is polynomial in $|x|$, and the reason why is as follows. First, we have $|y| \leq p(|x|)$, and therefore the length of the string $\langle x, y \rangle$ is polynomially bounded (in the length of x). Now, because $B \in \text{P}$, we have that membership in B can be tested in polynomial time. Because the input in this case is $\langle x, y \rangle$, this means that the time required to test membership in B is polynomial in $|\langle x, y \rangle|$. However, because the composition of

two polynomially bounded functions is another polynomially bounded function, we find that the time required to test whether $\langle x, y \rangle \in B$ is polynomial in the length of x. Step 3 can also be performed in time polynomial in the length of x, as can the test $|y| > p(|x|)$ in step 4.

Finally, again using the assumption that f is polynomially bounded, so that $f(n) = O(n^k)$ for some positive integer k, we find that the total number of times the steps just considered are executed is at most

$$2^{p(n)+1} - 1 = O\Big(2^{n^{k+1}}\Big). \tag{20.8}$$

Using the rather coarse upper-bound that every polynomially bounded function g satisfies $g(n) = O(2^n)$, we find that the entire computation of M runs in time

$$O\Big(2^{n^{k+2}}\Big). \tag{20.9}$$

We have established that M runs in exponential time, so $A \in \mathrm{EXP}$.

Now we know that

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}, \tag{20.10}$$

and we also know that

$$\mathrm{P} \subsetneq \mathrm{EXP} \tag{20.11}$$

by the time-hierarchy theorem. Of course this means that one (or both) of the following proper containments must hold: (i) $\mathrm{P} \subsetneq \mathrm{NP}$, or (ii) $\mathrm{NP} \subsetneq \mathrm{EXP}$. Neither one has yet been proved, and a correct proof of either one would be a major breakthrough in complexity theory. Indeed, determining whether or not $\mathrm{P} = \mathrm{NP}$ is viewed by many as being among the greatest unsolved mathematical challenges of our time.

20.2 Polynomial-time reductions and NP-completeness

We discussed reductions in Lecture 17 and used them to prove that certain languages are undecidable or non-semidecidable. *Polynomial-time reductions* are defined similarly, except that we add the condition that the reductions themselves must be given by polynomial-time computable functions.

Definition 20.4. Let Σ and Γ be alphabets and let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be languages. It is said that A *polynomial-time reduces* to B if there exists a polynomial-time computable function $f : \Sigma^* \to \Gamma^*$ such that

$$w \in A \Leftrightarrow f(w) \in B \tag{20.12}$$

for all $w \in \Gamma^*$. One writes

$$A \leq_m^p B \tag{20.13}$$

to indicate that A polynomial-time reduces to B, and any function f that establishes that this is so may be called a *polynomial-time reduction* from A to B.

Polynomial-time reductions of this form are sometimes called *polynomial-time mapping reductions* (and also *polynomial-time many-to-one reductions*) to differentiate them from other types of reductions that we will not consider—but we will stick with the term *polynomial-time reductions* for simplicity. They are also sometimes called *Karp reductions*, named after Richard Karp, one of the pioneers of the theory of NP-completeness.

With the definition of polynomial-time reductions in hand, we can now define NP-completeness.

Definition 20.5. Let Σ be an alphabet and let $B \subseteq \Sigma^*$ be a language.

1. It is said that B is NP-hard if $A \leq_m^p B$ for every language $A \in \text{NP}$.
2. It is said that B is NP-complete if B is NP-hard and $B \in \text{NP}$.

The idea behind this definition is that the NP-complete languages represent the hardest languages to decide in NP; *every* language in NP can be polynomial-time reduced to an NP-complete language, so if we view the difficulty of performing a polynomial-time reduction as being negligible, the ability to decide any one NP-complete language would give us a key to unlocking the computational difficulty of the class NP in its entirety. Figure 20.2 illustrates the relationship among the classes P and NP, and the NP-hard and NP-complete languages, under the assumption that $\text{P} \neq \text{NP}$.

Here are some basic facts concerning polynomial-time reductions and NP. For all of these facts, it is to be assumed that A, B, and C are languages over arbitrary alphabets.

1. If $A \leq_m^p B$ and $B \leq_m^p C$, then $A \leq_m^p C$.
2. If $A \leq_m^p B$ and $B \in \text{P}$, then $A \in \text{P}$.
3. If $A \leq_m^p B$ and $B \in \text{NP}$, then $A \in \text{NP}$.
4. If A is NP-hard and $A \leq_m^p B$, then B is NP-hard.
5. If A is NP-complete, $B \in \text{NP}$, and $A \leq_m^p B$, then B is NP-complete.
6. If A is NP-hard and $A \in \text{P}$, then $\text{P} = \text{NP}$.

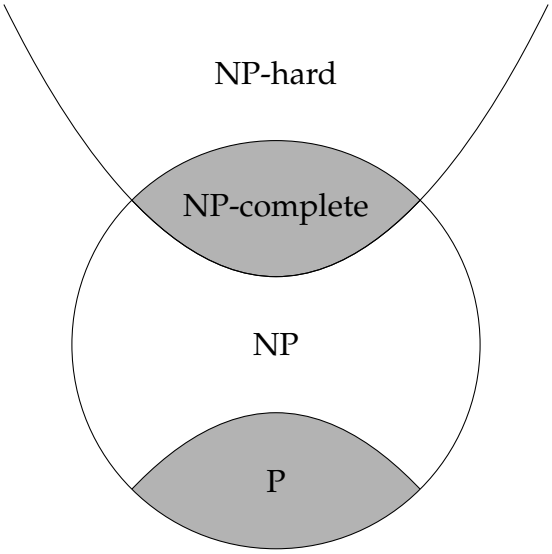

Figure 20.2: The relationship among the classes P and NP, and the NP-hard and NP-complete languages. The figure assumes P $\neq$ NP.

We typically use statement 5 when we wish to prove that a certain language B is NP-complete: we first prove that $B \in$ NP (which is often easy) and then look for a known NP-complete language A for which we can prove $A \leq_m^p B$.

The proofs of the statements listed above are all fairly straightforward, and you might try proving them for yourself if you are interested. Let us pick just one of the statements and prove it.

Proposition 20.6. *Let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be languages, for alphabets Σ and Γ, and assume $A \leq_m^p B$ and $B \in$ NP. It is the case that $A \in$ NP.*

Proof. Let us begin by gathering some details concerning the assumptions of the proposition.

First, because $A \leq_m^p B$, we know that there exists a polynomial-time computable function $f : \Sigma^* \to \Gamma^*$ such that

$$x \in A \;\Leftrightarrow\; f(x) \in B \tag{20.14}$$

for all $x \in \Sigma^*$. Because f is polynomial-time computable, there must exist a polynomially bounded function p such that $|f(x)| \leq p(|x|)$ for all $x \in \Sigma^*$.

Second, by the assumption that $B \in$ NP, there exists a polynomially bounded function q and a language $C \in$ P for which

$$B = \Big\{ x \in \Gamma^* \,:\, \exists y \in \{0,1\}^* \text{ such that } |y| \leq q(|x|) \text{ and } \langle x, y \rangle \in C \Big\}. \tag{20.15}$$

The DTM M operates as follows on input $w \in \Sigma^*$:

1. If w does not take the form $w = xx$ for some string $x \in \Sigma^*$, then reject.
2. Accept if $x \in A$, otherwise reject.

Figure 20.3: The DTM M from the proof of Corollary 20.7.

Now, define a new language

$$D = \{\langle x, y\rangle \,:\, \langle f(x), y\rangle \in C\}. \tag{20.16}$$

It is evident that $D \in \mathrm{P}$ because one may simply compute $\langle f(x), y\rangle$ from $\langle x, y\rangle$ in polynomial time (given that f is polynomial-time computable), and then test if $\langle f(x), y\rangle \in C$, which requires polynomial time because $C \in \mathrm{P}$.

Finally, observe that

$$A = \Big\{x \in \Sigma^* \,:\, \exists y \in \{0,1\}^* \text{ such that } |y| \leq q(p(|x|)) \text{ and } \langle x, y\rangle \in D\Big\}. \tag{20.17}$$

As the composition $q \circ p$ is a polynomially bounded function and $D \in \mathrm{P}$, it follows that $A \in \mathrm{NP}$. □

Let us conclude the lecture with the following corollary, which is meant to be a fun application of the previous proposition along with the time-hierarchy theorem.

Corollary 20.7. $\mathrm{NP} \neq \mathrm{DTIME}(2^n)$.

Proof. Assume toward contradiction that $\mathrm{NP} = \mathrm{DTIME}(2^n)$. Let Σ be any alphabet, let $A \subseteq \Sigma^*$ be an arbitrarily chosen language in $\mathrm{DTIME}(4^n)$, and define

$$B = \{xx \,:\, x \in A\}.$$

First we observe that $B \in \mathrm{DTIME}(2^n)$. In particular, the DTM M described in Figure 20.3 decides B in time $O(2^n)$. The reason why M runs in time $O(2^n)$ is as follows: the first step can easily be performed in polynomial time, and the second step requires $O(4^{n/2}) = O(2^n)$ steps, as A can be decided in time $O(4^m)$ on inputs of length m, and here we are deciding membership in A on a string of length $m = n/2$. The running time of M is therefore $O(2^n)$. As we have assumed that $\mathrm{NP} = \mathrm{DTIME}(2^n)$, it follows that $B \in \mathrm{NP}$.

Now define a function $f : \Sigma^* \to \Sigma^*$ as

$$f(x) = xx$$

for all $x \in \Sigma^*$. The function f can easily be computed in polynomial time, and it is immediate from the definition of B that

$$x \in A \Leftrightarrow f(x) \in B.$$

We therefore have that $A \leq_m^p B$. By Proposition 20.6, it follows that $A \in \text{NP}$, and given the assumption $\text{NP} = \text{DTIME}(2^n)$, it follows that $A \in \text{DTIME}(2^n)$.

However, as A was an arbitrarily chosen language in $\text{DTIME}(4^n)$, we conclude that $\text{DTIME}(4^n) \subseteq \text{DTIME}(2^n)$. This contradicts the time hierarchy theorem, so our assumption $\text{NP} = \text{DTIME}(2^n)$ was incorrect. □