INFR11102: Computational Complexity 01/10/2019

Lecture 5: Non-determinism

Lecturer: Heng Guo

1 Non-determinism

Last time we defined polynomial-time P as the class for "efficient" computation. Polynomial-time is "robust" in many senses. For example, unlike linear-time, it is closed by taking subroutines.

Another important notion regarding the model of computation is called *non-determinism*. However, we will define NP in a non-traditional (but equivalent) way first. Let us motivate it by the following example. Recall that a proper colouring of a graph is one where no edge is monochromatic.

Name: 3-COL

Input: A graph $G = (V, E)$.

Output: Is G 3-colourable?

The obvious algorithm is to enumerate all possible colourings, which would take $O(m3^n)$ time, where $n = |V|$ and $m = |E|$. Moreover, enumerating all colourings only requires $O(n)$ space, as we can erase the previous one once we move on to the next. There are faster algorithms, but no polynomial-time one is known. However, there is also no proof that it does not have one. Namely, it is open whether 3-COL $\in$ P? On the other hand, if we are given the graph G, and a colouring $\sigma : V \to \{0, 1, 2\}$, then we can easily verify whether this colouring σ is valid — we simply only check whether every edge is monochromatic.

Definition 1. *A language L is in the class* NP *if and only if there exists a deterministic polynomial-time TM M (called the verifier) and a polynomial $p(\cdot)$ such that*

1. ***Completeness**: if $x \in L$, there exists y such that $|y| \le p(|x|)$ and $M(x, y) = 1$;*

2. ***Soundness**: if $x \notin L$, then for any y such that $|y| \le p(|x|)$, $M(x, y) = 0$.*

Such a y is called the certificate.

Clearly, 3-COL $\in$ NP. In fact there are thousands of problems that are in NP but are still not known to be in P. However, we know that if 3-COL $\in$ P, then P = NP. This is captured by the so-called "NP-completeness", which we will cover later. The question whether P $\stackrel{?}{=}$ NP is the most important problem in computer science.

The essence of NP is that we can efficiently *verify* the solutions. However, for a problem to be in P, we need to be able to efficiently *find* the solution!

Here is another example for this verification vs. searching issue. Find an integer solution to $x^3 + y^3 + z^3 = C$ for various integers C. Only very recently we have a positive answer for all $C \leq 100$. On Sep 6th 2019, Booker and Sutherland found that

$$42 = (-80538738812075974)^3 + 80435758145817515^3 + 12602123297335631^3,$$

which is the last unsolved case for $C \leq 100$. Furthermore, on Sep 17th 2019, they found that

$$3 = 569936821221962380720^3 + (-569936821113563493509)^3 + (-472715493453327032)^3,$$

which is the third solution for $C = 3$ next to $(1, 1, 1)$ and $(4, 4, -5)$. Indeed, solutions mentioned above are the only ones up to 10^{16} due to their search.

When you have a problem that seems hard to solve, it is not always $\mathtt{NP}$-complete. Here is a non-trivial example. Let $G = (V, E)$ be a graph. A perfect matching (PM) is a subset $M \subseteq E$ of edges so that every vertex is adjacent to exactly one edge in M.

Name: PM

Input: A graph G.

Output: Does G have a perfect matching?

PM is indeed in $\mathtt{P}$! This was the original topic of Edmonds [Edm65], where he gave a polynomial-time algorithm to PM.

1.1 Non-deterministic Turing Machines

The traditional way of defining $\mathtt{NP}$ is via non-deterministic TMs (NTM). An NTM is the same as a deterministic one, except that there are more than one possible moves at each step, and an input is accepted if and only if there is a sequence of valid moves leading towards the accepting state.

In other words, the configuration graph $G_{M,x}$ for a TM M has out degree 1 for all vertices/configurations, whereas if M is NTM, then the out degree is not necessarily 1.[1] For an NTM N on input x, x is accepted if and only if there exists a path from q_0 to q_{acc} in $G_{N,x}$.

Similar to deterministic complexity classes, we may define non-deterministic complexity classes, such as $\mathtt{NTime}[f(n)]$ and $\mathtt{NSpace}[f(n)]$, for languages that can be computed by NTMs in $O(f(n))$ time or $O(f(n))$ space. An alternative way of defining $\mathtt{NP}$ is the following:

$$\mathtt{NP} := \bigcup_{c \in \mathbb{N}} \mathtt{NTime}[n^c].$$

Why are these two definitions equivalent? If $L \in \mathtt{NP}$ by some NTM N, then we construct the verifier M in Definition 1 by simulating N and treat y as the non-deterministic choices

[1] In fact, we may assume that the out degree is always 2. This is because we can simply simulate a k-way choice by a simple binary tree.

of N. Clearly y is at most polynomially long. If x is accepted by N, then such y must exist, and if x is not, then y does not exist.

Conversely, if L has a verifier M, then we can construct an NTM N by simulating M on just one input. Whenever M reads y, we list all possible choices of M in N by a non-deterministic move.

Similar to $\mathtt{NP}$, we define

$$\begin{aligned}\mathtt{NL} &:= \mathtt{NSpace}[\log n];\\ \mathtt{NPSpace} &:= \bigcup_{c\in\mathbb{N}} \mathtt{NSpace}[n^c];\\ \mathtt{NExp} &:= \bigcup_{c\in\mathbb{N}} \mathtt{NTime}[e^{n^c}].\end{aligned}$$

Since TM is a special case of NTM, we have that for any function $f(\cdot)$,

$$\begin{aligned}\mathtt{DTime}[f(n)] &\subseteq \mathtt{NTime}[f(n)];\\ \mathtt{DSpace}[f(n)] &\subseteq \mathtt{NSpace}[f(n)].\end{aligned}$$

Recall from the last lecture

$$\mathtt{DSpace}[S(n)] \subseteq \bigcup_{c\in\mathbb{N}} \mathtt{DTime}[2^{cS(n)}]. \tag{1}$$

We can strengthen (1) that

$$\mathtt{NSpace}[S(n)] \subseteq \bigcup_{c\in\mathbb{N}} \mathtt{DTime}[2^{cS(n)}],$$

by essentially the same argument — we construct the configuration graph and check whether there is a path to the accepting state. This gives $\mathtt{NL} \subseteq \mathtt{P}$ and $\mathtt{NPSpace} \subseteq \mathtt{Exp}$. Moreover,

$$\mathtt{NTime}[f(n)] \subseteq \mathtt{DSpace}[f(n)],$$

since, once again, we can construct the configuration graph and check the existence of an accepting path in the $O(f(n))$ space. To summarize, we have the following relationship among these complexity classes:

$$\mathtt{L} \subseteq \mathtt{NL} \subseteq \mathtt{P} \subseteq \mathtt{NP} \subseteq \mathtt{PSpace} \subseteq \mathtt{NPSpace} \subseteq \mathtt{Exp} \subseteq \mathtt{NExp}. \tag{2}$$

Note the containment $\mathtt{NP} \subseteq \mathtt{PSpace}$ is not obvious. However, this is correct, since, unlike the $\mathtt{P}$ vs. $\mathtt{NP}$ problem, we actually know that $\mathtt{PSpace} = \mathtt{NPSpace}$. It is known as Savitch's theorem, which we will cover next time. Unfortunately, this is pretty much the only thing we know stronger than (2).

Remark (Bibliographic). Relevant chapters are [AB09, Chapter 1] and [Pap94, Chapter 9].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17(3):449–467, 1965.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.