CSE200 Lecture Notes – NP

Lecture by Russell Impagliazzo
Notes by Jiawei Gao

February 2, 2016

The class $\mathsf{EXP} = \bigcup_k \mathsf{TIME}(2^{n^k})$ strictly contains $\mathsf{P} = \bigcup_k \mathsf{TIME}(n^k)$, by the Time Hierarchy Theorem. For some problems between P and EXP, we do not know whether they are in P, but we do not expect them to require exponential time.

Example: Factoring problem. Given N, find all $N = p_1^{e_1} \cdot \dots \cdot p_k^{e^k}$, where p_i are distinct primes. The input is in binary, therefore N is exponential in the input size.

Example: Information security. Legitimate users know some keys but attackers don't. If the attackers do exhaustive search on all possible security keys, it requires exponential time.

Example: Learning. checks if data satisfies some pattern. If we can guess a pattern, we can efficiently check if data satisfies this pattern.

1 Searching, optimization and decision problems

We define a class of searching problems.

Definition 1.1. SearchP is the class of *searching problems* satisfying:

- Instance: x.
- Solution format: y, $|y| = \mathrm{poly}(|x|)$.
- Constraints: $R(x, y) \in \mathsf{P}$.

Example: Factoring

- Instance: N.
- Solution format: $(p_1, e_1), \dots, (p_k, e_k)$, primes and exponents.
- Constraints: Each p_i is a prime, $N = p_1^{e_1} \cdot \dots \cdot p_k^{e_k}$.

Example: Graph 3-coloring

- Instance: $G = (V, E)$.
- Solution format: $\chi : V \to \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$.

- Constraints: If $\{u, v\} \in E$, then $\chi(u) \neq \chi(v)$.

Which of the two problems above is harder? We can prove that if 3-coloring is hard, it does not mean Factoring is also hard. So in some sense, 3-coloring is harder than Factoring.

Even if most problems in computer science are searching problems, the class NP is defined as a set of decision problems. We show that if we can solve the decision version in polynomial time, then we can solve the searching version in polynomial time.

Definition 1.2. NP is the class of *decision problems* satisfying:

- Instance: x
- Solution format: y, $|y| = \mathsf{poly}(|x|)$
- Constraints: $R(x, y) \in \mathsf{P}$
- Goal: Is there a y so that $R(x, y)$? Otherwise, output "Impossible".

Definition 1.3. OptP is the class of *optimization problems* satisfying:

- Instance: x
- Solution format: y
- Objective function: $F(x, y) \rightarrow \mathbb{Z} \in \mathsf{P}$.
- Goal: Given x, find $\arg\max_y F(x, y)$ (or $\arg\min_y F(x, y)$)

By doing brute-force search on y, we can solve these problems in exponential time. So $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$. (We do not know whether the containments are proper. And we conjecture both containments are proper.)

Next, we show that searching, optimization and decision problems are polynomial-time reducible to each other.

Theorem 1.1.

$$\mathsf{P} = \mathsf{NP} \iff \mathsf{P} = \mathsf{SearchP} \iff \mathsf{P} = \mathsf{OptP}$$

We show $\mathsf{P} = \mathsf{NP} \Longrightarrow \mathsf{P} = \mathsf{SearchP} \Longrightarrow \mathsf{P} = \mathsf{OptP} \Longrightarrow \mathsf{P} = \mathsf{NP}$. The first implication will be proved in section 1.1, and the second in section 1.2, and the third in section 1.3.

1.1 From searching to decision

For the 3-coloring problem, we can decide the coloring of each vertex one by one: first decide if the graph is 3-colorable given vertex v_1 is $color_1$, then decide if the graph is 3-colorable given vertex v_1 is $color_1$ and v_2 is $color_2$... until the graph is fully colored.

We generalize 3-coloring to allowing partial solutions.

Problem: Gen 3-coloring

- Instance: G, and a partial 3-coloring χ_0 of $V_0 \subseteq V$.
- Solution: $\chi : V \to \{\text{R}, \text{G}, \text{B}\}$.
- Constraints: In addition to being a 3-coloring, χ has to agree with χ_0 on colored positions V_0.
- Goal: Is there such a χ?

We can extend this technique to all NP problems. If we can solve a generic decision problem, we can solve a generic search problem.

Problem: EPS (extend partial solution)

- Instance: $x, y_1, \ldots, y_t$.
- Solution format: y, $|y| = \text{poly}(|x|)$.
- Constraints: $R(x, y) \in \mathsf{P}$, first t bits of y equals $y_1, \ldots, y_t$.
- Goal: Is there a y so that $R(x, y)$ where first t bits of y equals $y_1, \ldots, y_t$?

To solve the a searching problem, we can query EPS (a decision problem) polynomial times.

Algorithm: SolveSearch
<u>Instance: x</u>
$\ell \leftarrow$ length of solution
$t \leftarrow 0$
If EPS$(x, \lambda) = F$ return “Impossible”.
While $t < \ell$ do:
    If EPS$(x, y_1, \ldots, y_t, 0)$ then $t \leftarrow t + 1$, $y_{t+1} \leftarrow 0$
        else $t \leftarrow t + 1, y_{t+1} \leftarrow 1$

The algorithm queries EPS for $O(\ell)$ times, with queries no more than $n + \ell$ size.

Thus, $\mathsf{P} = \mathsf{NP} \Longrightarrow \text{EPS} \in \mathsf{P} \Longrightarrow \mathsf{SearchP} \subseteq \mathsf{P} \Longrightarrow \mathsf{SearchP} = \mathsf{P}$.

1.2 From optimization to searching

A maximization problem can be reduced to deciding if there is a solution of size at least k. For example, for the Max Independent Set problem, we can solve it by solving the Big Independent Set problem.

Problem: Max Independent Set

- Instance: G

- Solution format: $S \subseteq V$
- Constraints: For all $\{u, v\} \in E$, $u \notin S$ or $v \notin S$
- Goal: maximize $|S|$.

Problem: Big Independent Set

- Instance: G, k
- Solution format: $S \subseteq V$
- Constraints: For all $\{u, v\} \in E$, $u \notin S$ or $v \notin S$, and $|S| \geq k$.
- Goal: Is there such an S?

To solve Max Independent Set, we could do a linear search on the set size k, since $1 \leq k \leq n$. However, in general, the values for an optimization problem may be exponential in the input size. So instead, we can do a binary search.

One observation is that $|F(x, y)| \leq q(|x|)$. Thus, $-q(|x|) \leq F(x, y) \leq q(|x|)$. We use binary search to find $\arg\max F(x, y)$. The number of queries to the decision problem is at most $\log(2 \cdot 2^{q(|x|)}) = q(|x|) + 1$.

So $\mathsf{P} = \mathsf{SearchP} \implies \mathsf{P} = \mathsf{OptP}$.

1.3 From decision to optimization

This step is straightforward. For a decision problem, we create the following optimization problem.

Problem: Opt

- Instance x.
- Solution: y.
- Constraints: $F(x, y) = 1$ if $R(x, y)$, -1 otherwise.
- Goal: Find $y = \arg\max F(x, y)$. If $F(x, y) = 1$ output “Yes”, otherwise output “No”.

2 Nondeterministic computation

The class name “NP” means “Nondeterministic polynomial time”.

Recall that a deterministic Turing machine performs a *unique* action for every state and contents under tape head, While a nondeterministic Turing machine can perform *a set of possible actions* at each step. Therefore, an NTM has a set of runs on any one input. We view NTM N as accepting x if *any* run of N on x halts and accepts.

Theorem 2.1. $L \in \mathsf{NP}$ iff there is a nondeterministic TM N that recognizes L and runs in polynomial time on every run.

Proof.

- Let L be a language in NP, where L is defined as $x \in L \iff \exists y R(x, y)$. Then we can create an NTM that recognizes L. The NTM nondeterministically selects every bit of y. Finally the NTM deterministically checks if the relation R holds.
- Let N be a polynomial-time NTM. Let solution y be the sequence of nondeterministic moves N makes on x. We define $R(x, y)$ so that it simulates N on x, using y to choose which nondeterministic moves to take.

$\square$