Lecture 2: **P** vs. **NP**, and Polynomial-Time Hierarchy

Lecturer: Kurt Mehlhorn & He Sun

1 P vs. NP

Today's lecture starts with the intriguing question about **P** vs. **NP**. In early days of computer science, people discovered two families of problems which present different properties. Typical problems in the first family include:

- Given a graph G, is there a closed cycle which visits every edge of G exactly once?
- Given a graph G and two vertices s and t, what is the maximum flow between s and t?
- Given a graph G and two vertices s and t, find the shortest path between s and t.

All problems in this family can be solved in polynomial number of *steps*, by using either navïe methods, general algorithm design techniques (e.g. dynamic programming, greedy strategy, linear programming), or based on some mathematical properties of the problems (e.g. existence of an Euler cycle in a graph). On the other hand, there is a long list of problems where efficient algorithms are unknown. Typical problems include:

- `Clique`$_k$`:` Given a graph G of n vertices and a parameter k, is there a clique of size k in G?
- `Hamiltonian:` Given a graph G of n vertices, is there a cycle of length n which visits every vertex exactly one?
- `Satisfiability:` Given a boolean formula φ with n variables and m clauses, where every clause has at least three literals, is φ satisfiable? I.e. is there an assignment of $x_1, \ldots, x_n$ such that $\varphi(x_1, \ldots, x_n) = 1$?

While efficient algorithms were missing, these problems have some properties in common: (1) All existing algorithms for solving any problem in this family are essentially based on *Brute-Force Search*, i.e. listing all possible *candidate* solutions and checking if one candidate is a solution. (2) For every candidate solution, verifying the candidate is efficient and can be done in polynomial-time. Besides the problems listed above, the family includes many problems arising from Physics, Chemistry, Biology, as well as Network Design, City Planing, and etc.

The question of whether brute-force search can be avoided in general was discussed initially in a letter by Kurt Gödel. In 1956, Kurt Gödel wrote a letter to von Neumann. Despite that Kurt Gödel described the question in a remarkably modern way, the question can be more or less summarized by *how much we can improve upon the brute-force search.*

To start our formal discussion, we first relate *searching problems* to *decision problems.* Despite that most problems we handle are searching problems, i.e. finding a solution (an assignment of a boolean formula, an Euler cycle of a graph), we can simply study their decision version without decreasing the hardness of the problems. Given this, we relate the problem like *finding an assignment of boolean formula* φ to the problem like *is* φ *satisfiable?*

We further relate problems into *languages* or *sets*, such that for any instance (boolean formulae, graphs), the instance is in the set iff the answer to the instance is yes. For example, we rewrite the second and the third problem as

- `Hamiltonian`$\triangleq \{G :$ graph G has a Hamiltonian cycle$\}$
- `SAT`$\triangleq \{\varphi : \varphi$ is satisfiable$\}$

In order to show a specific graph (or formula) in `Hamiltonian` or `SAT`, we need a *certificate*, e.g. a permuation of n vertices (or an assignment of n boolean variables).

No we define **P** and **NP**. The complexity class **P** consists of the problems that can be solved in polynomial-time. The complexity class **NP** consists of all problems that admit a short "certificate" for membership. Given this certificate, called a witness, membership can be verified efficiently in polynomial-time. For instance, for a satisfiable formula $\varphi \in$ `SAT`, a true assignment is a certificate.

Since every polynomial-time solvable problem can be verified in polynomial-time, hence $\mathbf{P} \subseteq \mathbf{NP}$. However, because of various practical problems for which only brute-force based algorithms are known, people are interested in finding polynomial-time algorithms for such problems. The question about **P** vs. **NP** was precisely formulated in Cook's 1971 paper. Moreover, in that paper Cook showed that in order to prove $\mathbf{P} = \mathbf{NP}$, it suffices to study a subset of problems in **NP**. This subset consists of the "hardest" problems in **NP**, and is called **NPC**. Cook showed that a polynomial-time algorithm for any problem in **NPC** implies $\mathbf{P} = \mathbf{NP}$.

Theorem 1. *`SAT` is* **NP***-complete.*

The question about the **P** vs. **NP** can be informally formulated by the following question: *Is every polynomial-time verifiable problem solvable in polynomial-time?* Although most people believe that finding a solution is much more difficult than verifying the correctness of a solution and a lot of effort was made for proving $\mathbf{P} \neq \mathbf{NP}$ over the past 40 years, people have not found a promising way to prove or disprove this statement.

Why do we care the **P** vs. **NP** question? Among various consequence, we would like to mention a few here.

- $\mathbf{P} = \mathbf{NP}$ implies that every efficiently verifiable problem can be solved efficiently, and essentially brute-force search can be avoid. We are happy with the situation that every "hard" problem can be solved easily. However, as we will discuss in later lectures, based on this statement there is no modern cryptography.
- $\mathbf{P} \neq \mathbf{NP}$ implies that brute-force search is essential and can not be avoided for most problems. However, this negative result shows the existence of hard problems which are essential for cryptography.

So far we have the following results: (1) $\mathbf{P} \subseteq \mathbf{NP}$, (2) $\mathbf{NPC} \subseteq \mathbf{NP} \setminus \mathbf{P}$ if $\mathbf{P} \neq \mathbf{NP}$. Now the question: Assuming that $\mathbf{P} \neq \mathbf{NP}$, is there a problem that is in an "intermediate" state between **P** and **NPC**? I.e. is $\mathbf{NP} \setminus \mathbf{P} \setminus \mathbf{NPC} = \emptyset$?

Theorem 2 (Ladner, 1975)**.** *If* $\mathbf{P} \neq \mathbf{NP}$*, then* $\mathbf{NP} \setminus \mathbf{P} \setminus \mathbf{NPC} \neq \emptyset$.

Ladner showed that if $\mathbf{P} \neq \mathbf{NP}$, then there are infinitely many levels of difficulties in $\mathbf{NP}$. However assuming that $\mathbf{P} \neq \mathbf{NP}$ we know very few candidates which "should" be non $\mathbf{NP}$-complete.

Example (Graph Isomorphism)**.** *Given two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, find an isomorphism between G_1 and G_2, i.e. a permutation $\varphi : V_1 \mapsto V_2$ such that $(u, v) \in E_1$ iff $(\varphi(u), \varphi(v)) \in E_2$.*

We remark that these "conjectured" $\mathbf{NP}$ non-complete problems play a special role in Theoretical Computer Science. We will show in next lectures how to use these problems to design Cryptography protocols.

2 Polynomial-Time Hierarchy

An oracle is a hypothetical device that would solve a computational program, free of charge. For instance, say we have a subroutine to multiply two matrices. After we create the subroutine, we don't have to think about how to multiple two matrices again, we simply think about it as a "black block" which always returns the correct answer.

Now assume that we have a program M, and program M uses another program A as a subroutine. Program M can invoke A as many times as it likes. We use M^A to represent such a program. Moreover, we denote $\mathbf{P}^A$ by the set of polynomial-time algorithms where each algorithm can use A as a subroutine and the runtime of A is assumed to be $O(1)$. Moreover, for any complexity classes $\mathbf{A}, \mathbf{B}$, we define $\mathbf{A}^{\mathbf{B}}$ as the set of problems which can be solved by using an algorithm in $\mathbf{A}$ that invokes another procedure in $\mathbf{B}$ as a subroutine.

Example. $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$.

So far we discussed two complexity classes: $\mathbf{P}$ and $\mathbf{NP}$. While most practical problems can be categorized into these two classes and most people in 1970s were interested in studying the relationship between $\mathbf{P}$ and $\mathbf{NP}$, a young computer scientist, Larry S-tockmeyer, and his supervisor Meyer started to think what the next step is. In the 1972's paper, they looked at the following problem.

Problem 3. *The set `MINIMAL` consists of all boolean formulas for which there is no shorter and equivalent boolean formula.*

We look at the complexity status of $\overline{\texttt{MINIMAL}}$. Note that for any formula φ and φ' of n variables, there is a short certificate (a specific assignment of $x_1, \ldots x_n$) if φ is not equivalent to φ'. Hence the question of testing $\varphi \not\equiv \varphi'$ is in $\mathbf{NP}$, i.e. there is an algorithm $\mathcal{A} \in \mathbf{NP}$ to test if $\varphi \not\equiv \varphi'$. Moreover, given algorithm $\mathcal{A}$ as an oracle, for every $\varphi \in \overline{\texttt{MINIMAL}}$, there is a short certificate which can be verified in polynomial-time. Although we cannot show if $\overline{\texttt{MINIMAL}}$ is in $\mathbf{P}$ or $\mathbf{NP}$, the discussion above shows that $\overline{\texttt{MINIMAL}} \in \mathbf{NP}^{\mathbf{NP}}$.

Stockmeyer further extended this idea and used an inductive way to categorize the difficulties of the problems.

Definition 4 (Polynomial-Time Hierarchy)**.** *Σ_i is a sequence of sets and will be defined inductively:*

1. $\Sigma_1 \triangleq \mathbf{NP}$

2. $\Sigma_{i+1} \triangleq \mathbf{NP}^{\Sigma_i}$

Moreover, let $\Pi_i \triangleq \mathrm{co}\Sigma_i$ *and* $\Delta_{i+1} = \mathbf{P}^{\Sigma_i}$.

Definition 5. $\mathbf{PH} \triangleq \cup_{i\geq 0}\Sigma_i$.

Theorem 6. *(1) If* $\Sigma_i = \Pi_i$, *then* $\mathbf{PH} = \Sigma_i$. *(2) If* $\Sigma_i = \Sigma_{i+1}$, *then* $\mathbf{PH} = \Sigma_i$.

Like complete problems in set **NP**, there are complete problems for every Σ_i and Π_i.

Problem 7. *Language* Σ_i-`SAT` *is a set of boolean formulae* φ *such that there are vectors* $\overrightarrow{x_i}$ *of boolean variables satisfying*

$$\exists\overrightarrow{x_1}\forall\overrightarrow{x_2}\cdots Q\overrightarrow{x_i}\varphi(\overrightarrow{x_1},\ldots,\overrightarrow{x_i}) = 1,$$

where $Q = \forall$ *if* i *is even, and* $Q = \exists$ *otherwise. Language* Π_i-`SAT` *is defined similarly.*

Theorem 8. Σ_i-`SAT` *is* Σ_i*-complete, and* Π_i-`SAT` *is* Π_i*-complete.*

3 Further Reading

Main Reference.

- Wikipedia entry “NP (complexity)”.
- Wikipedia entry “Polynomial hierarchy”, and [Sto76].

Other Reference. Michael Sipser gave an excellent survey about the **P** vs. **NP** problem [Sip92], where Gödel’s letter and the English translation are inside. For Larry Stockmeyer’s work, [For05] is a good summary.

On complete problems in Polynomial-Time Hierarchy, there is a list maintained by Marcus Schaefer, and Christopher Umans. Just google “Completeness in the Polynomial-Time Hierarchy A Compendium” to get the latest version.

Bibliography

[For05] Lance Fortnow. Beyond NP: the work and legacy of larry stockmeyer. In *STOC*, pages 120–127, 2005.

[Sip92] Michael Sipser. The history and status of the P versus NP question. In *STOC*, pages 603–618, 1992.

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.