Lecture I
INTRODUCTION TO COMPLEXITY THEORY

§1. FROM ALGORITHMS TO COMPLEXITY

We all have informal understanding of "computational complexity". It is based on our experience with programming, or more abstractly, the concept of algorithms. In these lectures, "complexity" always means "computational complexity".

An important step from algorithms to complexity theory is to realize that "(computational) problems" are now the objects of study, not the objects of our attempt to find solutions. What are the mathematical properties of these problems?

We begin along the lines of Papadimitriou's Book.

- THE CONCEPT OF A PROBLEM AND ITS COMPLEXITY

  We will look at 3 illustrative problems.

- REACHABILITY Problem

  Given graph $G = (V, E)$, $V = [1..n]$, is there a path from 1 to n?

  This is a decision problem.

  How to solve? Algorithms! Informally:

  > $S = \{1\}$
  > Unmark all nodes but node 1.
  > While S non-empty,
  >   remove i from S
  >   put each unmarked neighbors of
  > j into S, and mark them.
  >   If $j = n$, done.

  Algorithms is also a mathematical object of study. Much details missing: how is G represented? How do we choose i from S? How do we look at neighbors of i?

  Specific: if G is an adjacency matrix, then this is $O(n^2)$.

- Asymptotics

  Complexity functions $f : \mathbb{N} \to \mathbb{N}$. Implicitly, $f(n) = \max\{0, ceil(f(n))\}$. Also, big-Omega, Theta.

  Polynomial complexity functions: $f = O(n^c)$ for some $c > 0$.

  FACT: if f is polynomial then $f(n) = O(c^n)$ for any $c > 1$, but $f(n)$ is not $\Omega(c^n)$.

- Tractability Thesis

  THESIS: polynomial=tractable/feasible

  non-polynomial = untractable

  Discuss!

  - n^{10} is not feasible

  - Worst case or average?
  - Time or Space?
  - Robustness of concept
  - Useful theory results

  In any case, once we accept this, we have the notion of an "upper bound" of the complexity of REACHABILITY. But what is its "intrinsic complexity"?

- Max Flow Problem

  Given a network, $N = (V, E, s, t, c)$ where $c(i, j) \geq 0$ is capacity of $(i, j) \in E$.

  Flow in N is f such that
  a) $0 \leq f(i, j) \leq c(i, j)$
  b) inflow=outflow at each $i \neq s, t$.

  Value of f is net outflow from s (= net inflow into t).

  Problem: find f with max value.

  Optimization Problem!

  Can convert it into a decision problem.

  Observation: the optimization problem is polynomial iff the decision problem is.

  POLYNOMIAL SOLUTION: given flows f, f', let $Df = f' - f$.

  Note: $Df(i, j)$ can be negative, which can be regarded as positive flow from j to i. Moreover, this negative value has absolute value at most $f(i, j)$.

  View Df as a flow in a DERIVED network $DN(f)$.

  Then, f is optimum iff there can be no positive flow in DN.

  Checking for positive flow is equivalent to REACHABILITY!

```
ALGORITHM:
    Begin with the zero flow.
    STAGE:
        If f is current flow, and there is
            a REACHABILITY path from s to t,
            we "augment" flow along this path.
        Else, current flow is optimum
```

  If capacities are integers at most C, then there are at most nC stages. Since each stage is $O(n^2)$, this algorithm is $O(n^3C)$.

  Bad Example in fig.1-5 of book.

  Discussion: is this poly time?

  Alternative Algorithm: always augment along the path with fewest edges. Then CLAIM: each edge or its reverse can be a bottle neck in at most n stages. There are $O(n^2)$ edges and at most $O(n^2)$ work per stage. Hence overall, algorithm is $O(n^5)$.

- Traveling Salesman Problem (TSP)

  The above two problems about graphs have polynomial solutions (although, for Max Flow, it was not immediately obvious). We now introduce another graph problem for which we know no polynomial time solution.

  The input for TSP is a directed graph G with positive integer weights $w(e)$ on each edge e. If e is not an edge, the weight is assumed to be $+\infty$. A **tour** of G is a cyclic permutation π of all the vertices in

G, and represents a cycle C_π. The **cost** of the tour is the sum of the weights of all the $n-1$ edges in C_π. The TSP problem is to compute a tour of minimum cost.

Despite great efforts over a long period of time, no one has succeeded in producing a polynomial time algorithm. The natural conjecture is that this goal is impossible. If turns out that this conjecture, true or false, has wide spread implications which we will discover. For this, we must first turn to a systematic approach to complexity theory.

§2. Standard Complexity Theory

Complexity theory has developed a very special (stylised) notion of what is a computational problem. It is one of the simplest form and yet it already many of the essential complexity notions of interest.

- One begins (naively) by asking about the "intrinsic complexity" of a problem P. For example, P_0 is integer multiplication (asked by Cobham) Another problem P_1 is graph isomorphism.
- Complexity theory must deal with different kinds of domains: integers, graphs, sets, polynomials, etc. E.g., for integer multiplication, the domain is $D = \mathbb{N}^2$, and for P_2, D =graphs. To unify all these domains, we insist:

  (A) The problem input and output are encoded as strings over Σ.

  Important: we side-step the issue of how to encode a natural domain (integers, graphs, sets, etc) into Σ^*. That is considered part of the problem specification! E.g., numbers are usually assumed to be in binary notation.
- Again, what is a problem P? There are many forms of problems. The most obvious is where P is a function. Complexity theory will focus on the special case where the function P is $0/1$.

  (B) A problem P is a language $P \subseteq \Sigma$.

  EXAMPLE: We can usually convert a non-decision problem P into a decision problem P' whose complexity is polynomially related that of P. For the TSP problem above, we can consider a related problem: The Traveling Salesman Decision Problem (TSD). The input is the same as TSP, but another integer k is given. The problem is to decide whether there is a tour with cost at most k. It is an exercise to show that TSD is polynomially related to TSP.
- Because of (A), some strings in Σ^* may not be well-formed (does not represent elements in the natural domain of P). But we insist:

  (C) All strings in Σ^* are valid inputs for our problem.

  In other words, any algorithmic solution must know how to parse the input (and reject if the input is not well
- Once we have agreed on the problem, we need to agree on the computational model. E.g., if P is the sorting problem, there are still many VERY different models (comparison tree, comparator network, tape models).
- There are many models that have been studied over the years. Here are 3 important ones: Turing Machines, Pointer Machines, Random Access Machines (RAM).

  In classical computability theory, we have a powerful principle:

  CHURCH'S THESIS: the notion of computability is invariant across all (general) computational models.

- In complexity theory, the picture is much more complicated. To put some order into this mess, we must make two important observations that cuts across all "general" computational models:

  (1) COMPUTATIONAL MODES: Each computational model can operate in different modes: e.g., non-deterministic, probabilistic, parallel. EACH OF THESE MODES ARE SIGNIFICANT!

  (2) COMPUTATIONAL RESOURCES: Within each computational model, we can identify computational resources corresponding as TIME, SPACE and REVERSAL. (Occasionally, a model may be missing some of these.)

- We now have a modified version of CHURCH'S THESIS:

  (H) A problem is tractable if it has polynomial complexity.

  (I) In each general computational model, there are natural notions of time, space and reversal complexity resource.

  (J) For each computational mode, and each computational resource, the notion of intractability is invariant.

  Many basic questions of complexity relates to the tractable classes defined in (J). For example:

  (P versus NP Question) With respect to time resource, the notion of intractability in the fundamental mode is different from the notion of intractability in the non-deterministic mode.

§3. PRELIMINARIES

- Numbers: $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$.

- Complexity Function: these are partial functions from $\mathbb{R}$ to $\mathbb{R} \cup \{\infty\}$. Say $f \geq g(\text{ev.})$ if $f(x) \geq g(x)$ for $x \geq x_0$. Then $O(f)$ is the set of complexity functions g such that there is some $C = C(g) > 0$, $0 \leq g \leq Cf$.

- Languages: Σ is a finite set of symbols (alphabet). Σ^* are all finite strings over Σ. We let $|w|$ denote the length of a string. $\emptyset \in \Sigma^*$ denote the empty string.

  A language is a pair (Σ, L) where $L \subseteq \Sigma^*$. We usually simply call L the language. The complement of L is $\Sigma^* \setminus L$.

  If K is a collection of languages, co-K is the set of complements of languages in K.

- CONVENTION ALPHA: Σ_∞ is the universal set of symbols. Distinguished symbol: $\square$. Alphabets are finite subsets of Σ_∞, minus $\square$.

- CONVENTION BETA: Q_∞ is the universal set of states, disjoint from Σ_∞. Three distinguished states: q_0, q_a, q_r.