csce750 — Analysis of Algorithms
Fall 2020 — Lecture Notes: NP-Complete Problems

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 Introduction

CLRS 34
GJ 1–3

The theory of **NP-completeness** can be used cast doubt on the existence of *any* polynomial-time algorithm for a given problem.

- So far, we have concentrated mostly on designing and analyzing efficient algorithms for various problems. Algorithms are evidence of how **easy** those problems are.
- By showing that a problem is NP-complete, we are giving evidence of how **hard** a problem is.

Practically, we can think of an NP-completeness proof as a 'license' to stop looking for an efficient algorithm, and settle for approximation or to consider only special cases.

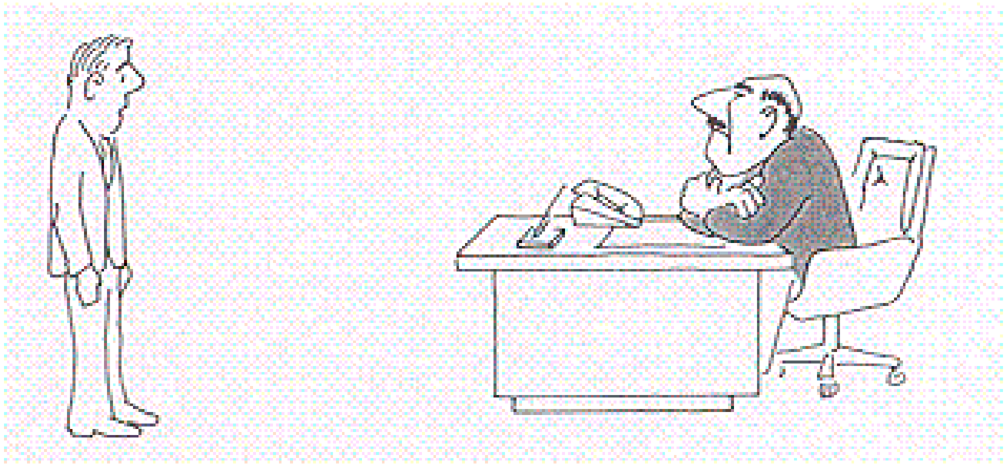
Note: Both CLRS and GJ define things at higher level of precision than we'll examine in the lecture: models of computation, abstract vs. concrete problems, encodings, *etc.*

2 Four classes of problems

We'll examine four different classes of problems.

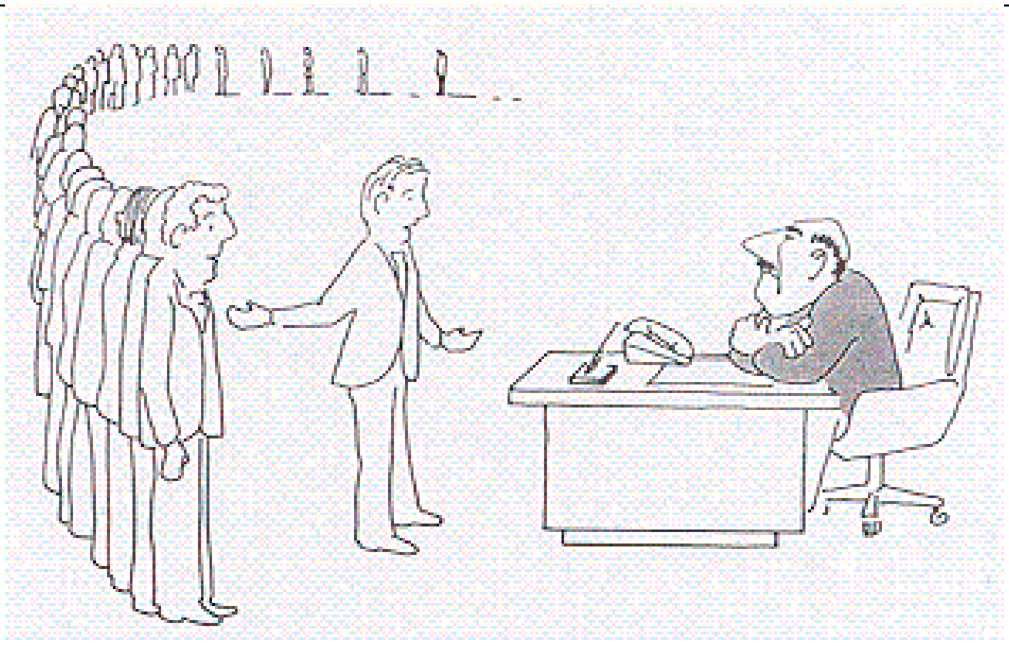
- P — problems that can be decided in polynomial time
- NP — problems that can be verified in polynomial time
- NP-hard — problems to which anything in NP can be reduced in polynomial time
- NP-complete — problems in both NP and NP-hard.

3 Illustration (from Garey and Johnson)



"I can't find an efficient algorithm. I guess I'm just too dumb."

"I can't find an efficient algorithm, because no such algorithm is possible."



"I can't find an efficient algorithm, but neither can any of these famous people."

4 Decision problems

A **decision problem** is a problem in which the correct output for each instance is either 'Yes' or 'No'.

HAM-CYCLE:
Instance: An undirected graph $G = (V, E)$.
Question: Does G contain a cycle that visits every vertex exactly once?

PATH:
Instance: A weighted directed graph G, a pair of vertices $u, v \in V(G)$, and a number k.
Question: Does G contain a path from u to v with total weight at most k?

5 Aside: Optimization problems

We can convert any **optimization problem** ('Find the largest...', 'Find the smallest...', etc) to a decision problem by including a bound on the objective function as part of the input.

6 Problems as languages

We can think of a decision problem as a **formal language**.

- The **input** is a (binary) string s.
- The **output** is either 'Yes' or 'No'.

The **language** of a problem is the set of binary input strings, under a suitable encoding, for which the correct output is 'Yes'.

7 *Example language*

HAM-CYCLE:
Instance: An undirected graph $G = (V, E)$.
Question: Does G contain a cycle that visits every vertex exactly once?

The language $L_{\text{HAM-CYCLE}}$ is the set of strings describing undirected graphs that have Hamiltonian cycles.

8 *Deciding a language*

Definition: An algorithm **decides a language** if it correctly determines whether its input string is a member of that language.

Specifically, the algorithm should:

- Terminate for any input instance.
- Return 'yes' if the input instance is in the language.
- Return 'no' if the input instance is not in the language.

9 P

Intuition: P is the set of all problems that can be decided in polynomial time.

Definition: P is the set of all languages L for which there exists an algorithm A and a constant c, such that

- A decides L, and
- the worst-case run time of A is $O(n^c)$.

10 *How to prove:* P

To prove that your problem L is in P:

- Describe an algorithm whose input is an instance of L.
- Show that your algorithm decides L.
  - If the input is a Yes instance, must return Yes.
  - If the input is a No instance, must return No.
- Show that the worst-case run time of your algorithm is bounded by some polynomial.

11 Verification algorithms

A **verification algorithm** accepts two inputs:

- An ordinary string x (the 'real' input)
- A **certificate** y (a 'proof' that the correct answer for x is Yes).

A verification algorithm produces one of two outputs:

- 'Yes', or
- 'No'.

The **language verified** by a verification algorithm is

$$L = \{x \mid \text{there exists } y \text{ for which } A(x, y) \text{ outputs Yes.}\}$$

12 Verification example: HAM-CYCLE

We can form a verification algorithm for HAM-CYCLE that accepts two inputs:

- An undirected graph $G = (V, E)$.
- An ordered list of vertices $(v_1, \ldots, v_m)$.

The algorithm would:

- Output 'Yes' if:
  - the sequence $(v_1, \ldots, v_m)$ contains all of the vertices of V, with no duplicates, and
  - E contains an edge (v_i, v_{i+1}) for each $i = 1, \ldots, m-1$.
  - E contains an edge (v_m, v_1).
- Output 'No' otherwise.

13 Verification example: PATH

We can form a verification algorithm for PATH that accepts two inputs:

- A weighted directed graph G, a pair of vertices $u, v \in E(G)$, and a number k.
- An ordered list of vertices $(v_1, \ldots, v_m)$.

The algorithm would:

- Output 'Yes' if:
  - $v_1 = u$,
  - $v_m = v$,
  - E contains an edge (v_i, v_{i+1}) for each $i = 1, \ldots, m$, and
  - the total weight of all these edges is at most k.
- Output 'No' otherwise.

14 NP

Intuition: NP is the set of all problems for which 'Yes' answers can be verified in polynomial time.

Definition: NP is the set of all languages L for which there exists a verification algorithm A and a constant c, such that

- A verifies L, and
- the worst-case run time of A is $O(n^c)$.

~~"non-polynomial time"~~ "nondeterministic polynomial time"

15 How to prove: NP

To prove that your problem L is in NP:

- Decide what to use as the certificate. That is, for each yes instance, explain what a certificate should be.
- Describe an algorithm whose input is an instance of L and a certificate.
- Show that your algorithm verifies L.
  - If it's a Yes instance and the certificate is correct, it returns Yes.
  - Otherwise, it returns No.
    - If it's a Yes instance but the certificate is not correct.
    - If it's a No instance.
- Show that the worst-case run time of your algorithm is bounded by some polynomial.

16 P $\subseteq$ NP

Every problem in P is also in NP.

Why?

17 NP $\subseteq$ P?

Are there problems in NP that are not in P?

That is: Are all problems that are polynomially verifiable also polynomially solvable?

Note: If NP $\subset$ P, then P $=$ NP.

18 Reductions

We can show relationships between problems using **reductions**:

Definition: A language L_1 is **polynomial-time reducible** to another language L_2 if there exists a polynomial time algorithm f such that

$$x \in L_1 \text{ if and only if } f(x) \in L_2$$

If L_1 is polynomial-time reducible to L_2, we write $L_1 \leq_P L_2$