Algorithms: Design and Analysis, Part II

NP-Completeness

Definition and Interpretation

The Class NP

Refined idea: Prove that TSP is as hard as all brute-force-solvable problems.

Definition: A problem is in NP if:

(1) Solutions always have length polynomial in the input size

(2) Purported solutions can be verified in polynomial time.

Examples: - Is there a TSP tour with length ≤ 1000?

- Constraint satisfaction problems (e.g., 3SAT)

Interpretation of NP-Completeness

Note: Every problem in NP can be solved by brute-force search in exponential time. [Just check every candidate solution.]

Fact: Vast majority of natural computational problems are in NP [$\approx$ Can recognize a solution]

By definition of completeness: A polynomial-time algorithm for one NP-complete problem solves <u>every</u> problem in NP efficiently [i.e., implies that P$=$NP]

Upshot: NP-completeness is strong evidence of intractability!

A Little History

Interpretation: An NP-complete problem encodes simultaneously all problems for which a solution can be efficiently recognized (a "universal problem").

Question: Can such problems really exist?

Amazing fact #1: [Cook '71, Levin '73] NP-complete problems exist.

Amazing fact #2: [started by Karp '72] 1000s of natural and important problems are NP-complete (including TSP).

NP-Completeness User's Guide

Essential tool in the programmer's toolbox: The following recipe for proving that a problem Π is NP-complete.

(1) Find a known NP-complete problem Π' (see e.g. Garey + Johnson, Computers + Intractability)

(2) Prove that Π' reduces to Π

$\Rightarrow$ implies that Π at least as hard as Π'

$\Rightarrow$ Π is NP-complete as well (assuming Π is an NP problem)