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 $\leq 1000$?

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

Tim Roughgarden

# 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
[≈ can recognize a solution]

**By definition of completeness:** a polynomial-time algorithm for one NP-complete problem solves every problem in NP efficiently
[i.e., implies that P=NP]

**Upshot:** NP-completeness is strong evidence of intractability!

Tim Roughgarden

# 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 $\Pi$ is NP-complete.

① find a known NP-complete problem $\Pi'$

↳ see e.g. Garey + Johnson, Computers + Intractability

② prove that $\Pi'$ reduces to $\Pi$

⇒ implies that $\Pi$ at least as hard as $\Pi'$

⇒ $\Pi$ is NP-complete as well (assuming $\Pi$ is an NP problem)