VIII. NP-completeness
NP-Completeness – Overview

1. Introduction
2. P and NP
3. NP-complete (NPC): formal definition
4. How to prove a problem is NPC
5. How to solve a NPC problem: approximate algorithms
1. Introduction

Tractable and intractable problems

- Problems that are solvable by polynomial-time algorithms are tractable.

- Problems that require superpolynomial time are intractable.

Almost all the algorithms we have studied thus far have been polynomial-time algorithms on inputs of size $n$, their worst-case running time is $O(n^k)$ for some constant $k$. 
1. Introduction

NP-complete (NPC) problems: an informal definition

A large class of very diverse problems share the following properties:

1. We *only know* how to solve those problems in time much larger than polynomial, namely exponential time.

2. If we could *solve one NPC problem* in polynomial time, then there is a way to *solve every NPC problem* in polynomial time.
1. Introduction

Reasons to study NPC problems – practical

- There is a large class of very diverse intractable problems, and the difference between tractable and intractable may appear “only slightly”;
- you can use a known algorithm for an intractable problem, and accept that it will take a long long time to solve;
- you can settle for approximating the solution, e.g., finding a nearly best solution rather than the optimum; or
- you can change your problem formulation so that it is solvable in polynomial time.
1. Introduction

Reasons to study NPC problems – theoretical

- We stated above that “*We only know*” how to solve those problems in time much larger than polynomial, *Not that we have proven* that these problems require exponential time.

- Indeed, this is one of the most famous problems in computer science:

\[
P \neq \text{NP}
\]

or

- *Whether NPC problems have polynomial solutions?*

- First posed in 1971
  - [http://www.claymath.org/millennium-problems](http://www.claymath.org/millennium-problems)
1. Introduction

P-vs-NP example 1.

- **Shortest path:** finding the shortest path from a single source in a directed graph.
- **Longest path:** finding the longest simple path between two vertices in a directed graph.

*The first one is solvable in polynomial time (the Bellman-Ford algorithm), and the second is NPC, but the difference appears to be slight.*
1. Introduction

P-vs-NP example 2.

- **Euler tour**: given a connected, directed graph $G$, is there a cycle that visits each edge exactly once (although it is allowed to visit each vertex more than once)?

- **Hamiltonian cycle**: given a connected directed graph $G$, is there a simple cycle that visits each vertex exactly once?

The first one is solvable in polynomial time\(^1\), and the second is NPC, but the difference appears to be slight.

\(^1\)Euler cycle of $G = (V, E)$ iff $\text{in-degree}(v) = \text{out-degree}(v)$ for $\forall v \in V$. 
1. Introduction

P-vs-NP example 3.

- **Minimum spanning tree (MST):**
  given a weighted graph and an integer $k$, is there a spanning tree whose total weight is $k$ or less?

- **Traveling salesperson problem (TSP):**
  given a weighted graph and an integer $k$, is there a cycle that visits all vertices exactly once whose total weight is $k$ or less?

*The first one is solvable in polynomial time (Prim's and Kruskal's algorithms), and the second is NPC, but the difference appears to be slight*
1. Introduction

Traveling salesperson problem (TSP)
1. Introduction

P-vs-NP example 4.

- **Circuit value:**
  given a Boolean formula and its input, is the output True?

- **Circuit satisfiability (SAT):**
  given a Boolean formula, is there a way to set the inputs so that the output is True?

*The first one is solvable in polynomial time, and the second is NPC, but the difference appears to be slight.*
1. Introduction

Optimization problems and Decision problems

- Most of problems occur naturally as optimization problems,
- but they can also be formulated as decision problems, that is, problems for which the output is a simple Yes or No answer for each input.

Remarks:

- To simplify discussion, we can consider only decision problems, rather than optimization problems.
- The optimization problems are at least as hard to solve as the related decision problems, we have not lost anything essential by doing so.
1. Introduction

Optimization-vs-Decision example 1.

*Graph coloring:* A coloring of a graph $G = (V, E)$ is a mapping

$$C : V \rightarrow S$$

where $S$ is a finite set of “colors”, such that

$$(u, v) \in E \Rightarrow C(u) \neq C(v)$$

- **optimization problem:** given $G$, determine the smallest number of colors needed.

- **decision problem:** given $G$ and a positive integer $k$, is there a coloring of $G$ using at most $k$ colors?
1. Introduction

Optimization-vs-Decision example 2.

*Hamiltonian cycle:* A Hamiltonian cycle is cycle that passes through every vertex exactly once.

- **decision problem:** Does a given graph have a Hamiltonian cycle?
- **optimization problem:** Give a list of vertices of a Hamiltonian cycle.
1. Introduction

Optimization-vs-Decision example 3.

*TSP (Traveling Salesperson Problem)*: given a weighted graph and an integer $k$, is there a cycle that visits all vertices exactly once (Hamiltonian cycle) whose total weight is $k$ or less?

- **optimization problem**: given a weighted graph, find a minimum Hamiltonian cycle.

- **decision problem**: given a weighted graph and an integer $k$, is there a Hamiltonian cycle with total weight at most $k$?
1. Introduction – recap

1. Tractable and intractable problems
   polynomial-boundness: $O(n^k)$
2. NP-complete problems – informal definition
3. P vs NP
   difference may appear “only slightly”
4. Optimization problems and decision problems