# Chapter 1

# Introduction to computational complexity

## 1.1 Introduction

**Motivation:** We want to understand *what* is a problem and *when* a problem is *easy/hard*.

**Our strategy:** An intuitive review of the basic ideas in Computational Complexity theory.

**Key concepts:**

- Problem types: optimization problems, decision problems, feasibility problems.
- Instance (of a problem)
- A problem is a collection of instances.
- Size of an instance
- Algorithm
- Running time of an algorithm; worst-case time complexity
- Complexity classes: $P$ and $NP$

## 1.2 Problem, instance, size

### 1.2.1 Problem, instance

**Definition 1** (Problem)**.** *Is a* **generic** *question/task that needs to be answered/solved. A problem is a "collection of instances" (see below).*

A *particular* realization of a problem is define next.

**Definition 2** (Instance)**.** *Is a specific case of a problem. In other words, we can say "Instance∈Problem".*

### 1.2.2 Format and examples of problems/instances

A problem is an abstract concept. We will write problems in the following format:

- **INPUT:** Generic data/instance.
- **OUTPUT:** Question to be answered and/or task to be performed with the data.

**Examples of problems/instances:**

- Typical problems: optimization problems, decision problems, feasibility problems.
- LP and IP feasibility, TSP, IP minimization, Maximum cardinality independent set.

### 1.2.3 Size of an instance

The size of an instance is the *amount of information* required to represent the instance (in the computer).

**Definition 3** (Binary size/length)**.** *Is the number of bits that are needed in order to give the problem to a computer.*

**Examples of sizes:**

- Size of an integer/rational number.
- Size of a rational matrix.
- Size of a graph (node-egde matrix represetation).

## 1.3 Algorithms, running time, Big-O notation

### 1.3.1 Basics

**Definition 4** (Algorithm)**.** *List of instructions to solve a problem.*

**Definition 5** (Running time of an algorithm)**.** *Is the number of steps (as a function of the size) that the algorithm takes in order to solve an instance.*

### 1.3.2 Worst-time complexity

Given an algorithm  to solve a problem $P$, the *running time of* , as a function of the size $\sigma \in \mathbb{Z}_+$ will be defined as follows:

- The (generic) running time will be a function $f : \mathbb{Z}_+ \to \mathbb{Z}_+$.
- Given $\sigma$, the function $f$ is defined as follows:

$$f(\sigma) = \max\{\text{running time of } \text{ for instance } z, \text{ where size}(z) \leq \sigma\}.$$

**Remark:** This is a very conservative/pessimistic measure of running time.

### 1.3.3 Big-O notation

A function $f : \mathbb{Z}_+ \to \mathbb{Z}_+$ belongs to the class of functions $O(g(n))$ (that is, $f \in O(g(n))$) if there exists $c > 0$, $n_0 \in \mathbb{Z}_+$ such that

$$f(n) \leq cg(n), \quad \text{for all } n \geq n_0.$$

We usually say "$f$ is $O(g(n))$" or "$f$ is order $O(g(n))$".

### 1.3.4 Examples

- Basic examples of Big-O notation: $O(1)$, $O(n^k)$, $O(c^n)$, $O(\log(n))$, etc.
- An illustration of the fact that the running time depends on the size of the instance: the algorithm for the binary knapsack problem that is $O(nb)$ is not polynomial, since the size of the instance is $\log(b)$.

## 1.4 Basics

**Definition 6** (Polynomial time algorithm)**.** *An algorith is said to be a* polynomial time algorithm *if its running time is* $O(n^k)$ *for some* $k \geq 1$ *(where* $n$ *representes the size of a generic instance).*

**Remark:** *Polynomial time algorithms* are also known as *Polytime algorithms*.

**Definition 7** (Decision problem)**.** *A decision problem is any problem whose only acceptable answers are either YES or NO (but not both at the same time).*

**Some examples:** feasibility problems, decision version of optimization problems, etc.

## 1.5 Complexity classes

We will introduced 3 complexity classes (For see at least 495 more classes, please see http://complexityzoo.uwaterloo.ca/Complexity_Zoo).

### 1.5.1 Polynomial time problems

**Definition 8** (The class $\mathcal{P}$)**.** *Is the set of all decision problems for which a YES or NO answer for a particular instance can be* **obtained** *in polytime.*

**Remark:** For a particular problem $P$, there are 3 possibilities: (1) $P \in \mathcal{P}$, (2) $P \notin \mathcal{P}$, and $P \; ? \; \mathcal{P}$ (i.e., we don't know).

**Examples:**

- Shortest path
- Max flow
- Min cut
- Matroid optimization
- Matchings
- Linear programming

### 1.5.2 Non-deterministic polynomial time problems

**Definition 9** (The class NP)**.** *Is the set of all decision problems for which a YES answer for a particular instance can be* **verified** *in polytime.*

**Examples:**

- All problems in $\mathcal{P}$
- Integer programming
- TSP
- Binary knapsack
- Maximum independent set
- Subset sum
- Partition
- SAT, $k$-SAT
- Clique

### 1.5.3 Complements of problems in NP

**Definition 10** (The class NP)**.** *Is the set of all decision problems for which a NO answer for a particular instance can be* **verified** *in polytime.*

**Examples:**

- All problems in $\mathcal{P}$
- Every "complement" of an NP problem
- PRIMES

**Remark**: Actually, PRIMES $\in$ NP (see *Pratt's certificates*), even better, it was recently proven that PRIMES $\in \mathcal{P}$ (by Manindra Agrawal, Neeraj Kayal, Nitin Saxena in 2004).

## 1.6 Relationship between the classes

### 1.6.1 A basic result

**Theorem 11.** *The following relationship holds:*

$$\mathcal{P} \subseteq NP \cap coNP.$$

### 1.6.2 An $1.000.000 open question

The question "Is $\mathcal{P} = \text{NP}$?" is one of the most important problems in mathematics and computer science. A correct answer is worth 1 Million dollars! Most people believe that $\mathcal{P} \neq \text{NP}$.

## 1.7 Comparing problems, Polynomial time reductions

**Motivation:** We would like to solve problem $P_1$ by efficiently *reducing* it to another problem $P_2$ (why? Perhaps we know how to solve $P_2$!!!).

**Definition 12** (Polynomial time reductions)**.** *Let $P_1, P_2$ be decision problems. We say that $P_1$ is* polynomially reducible to $P_2$ *(denoted $P_1 \leq_{\mathcal{P}} P_2$) if there exists a function*

$$f : P_1 \to P_2$$

*such that*

1. *For all $w \in P_1$ the answer to $w$ is YES if and only if the answer to $f(w)$ is YES.*

2. *For all $w \in P_1$ $f(w)$ can be computed in polynomial time w.r.t to* size$(w)$*. In particular, we must have that* size$(f(w))$ *is polynomially bounded by* size$(w)$*.*

**Remarks:**

1. In the above definition "$f$ efficiently transforms an instance of $P_1$ into an instance of $P_2$". In particular, if we know how to solve problem $P_2$, then we can solve problem $P_1$.

2. Therefore, the notation $P_1 \leq_{\mathcal{P}} P_2$ makes sense: we are saying that $P_1$ is "easier" to solve than $P_2$, as any algorithm for $P_2$ would work for $P_1$.

## 1.8 Comparing problems, Polynomial time reductions

### 1.8.1 Definition

**Motivation:** We would like to solve problem $P_1$ by efficiently *reducing* it to another problem $P_2$ (why? Perhaps we know how to solve $P_2$!!!).

**Definition 13** (Polynomial time reductions)**.** *Let $P_1, P_2$ be decision problems. We say that $P_1$ is* polynomially reducible to $P_2$ *(denoted $P_1 \leq_{\mathcal{P}} P_2$) if there exists a function*

$$f : P_1 \to P_2$$

*such that*

1. *For all $w \in P_1$ the answer to $w$ is YES if and only if the answer to $f(w)$ is YES.*

2. *For all $w \in P_1$ $f(w)$ can be computed in polynomial time w.r.t to* size$(w)$*. In particular, we must have that* size$(f(w))$ *is polynomially bounded by* size$(w)$*.*

**Remarks:**

1. In the above definition "$f$ efficiently transforms an instance of $P_1$ into an instance of $P_2$". In particular, if we know how to solve problem $P_2$, then we can solve problem $P_1$.

2. Therefore, the notation $P_1 \leq_{\mathcal{P}} P_2$ makes sense: we are saying that $P_1$ is "easier" to solve than $P_2$, as any algorithm for $P_2$ would work for $P_1$.

### 1.8.2 Basic properties

**Proposition 14.** *Let $P_1, P_2$ be two problems such that $P_1 \leq_{\mathcal{P}} P_2$ and assume that $P_2 \in \mathcal{P}$. Then $P_1 \in \mathcal{P}$.*

**Proposition 15.** *Let $P_1, P_2$ be two problems such that $P_1 \leq_{\mathcal{P}} P_2$ and assume that $P_2 \in NP$. Then $P_1 \in NP$.*

**Proposition 16.** *Let $P_1, P_2, P_3$ be three problems assume that $P_1 \leq_{\mathcal{P}} P_2$ and $P_2 \leq_{\mathcal{P}} P_3$. Then $P_1 \leq_{\mathcal{P}} P_3$.*

## 1.9 NP-Completeness

### 1.9.1 The basics

**Definition 17** (NP-Completeness)**.** *A decision problem $P$ is said to be NP-complete if:*

1. *$P \in NP$*
2. *$Q \leq_{\mathcal{P}} P$ for all $Q \in NP$ (that is, every problem $Q$ in NP can be polynomially reduced to $P$).*

**Proposition 18.** *If $P$ is NP-complete and $P \in \mathcal{P}$ then $\mathcal{P} = NP$.*

### 1.9.2 Do NP-complete problems exist?

**Theorem 19** (S. Cook, 1971)**.** *SAT is NP-complete.*

## 1.10 NP-Hardness

**Definition 20** (NP-Completeness)**.** *A problem $P$ is said to be NP-hard if there exists a NP-complete decision problem that can be reduced to it.*

**Remarks:**

- NP-complete problems are NP-hard.
- Problems in NP-hard not need to be decision problems.
- Optimization versions of NP-complete decision problems are NP-hard.
- If $P$ is NP-hard and $P \in \mathcal{P}$ then $\mathcal{P} = \text{NP}$.

## 1.11 Exercises

1. Let $P, Q$ be decision problems such that every instance of $Q$ is an instance of $P$ (that is $\{\text{instances in } Q\} \subseteq \{\text{instances in } P\}$).
   (a) Give an example of $P, Q$ such that $Q \in \mathcal{P}$ and $P \in \text{NP} - \text{complete}$.
   (b) Give an example of $P, Q$ such that $Q, P \in \text{NP} - \text{complete}$.

**Note:** You must prove that your problem belongs to the corresponding class unless we have proved or sketched the proof of that fact in class.

## Solution:

(a)
- Let $P$ be the *Knapsack problem*. Then $P$ is NP-complete (see Problem 5).
- Let $Q$ be the special case of the *Knapsack problem* where all weights are 1, that is, $a_1, \ldots, a_n = 1$. The following algorithm decides this special case:

ALGORITHM:

1. List the objects $1, \ldots, n$ in decreasing order according to $c_1, \ldots, c_n$.
2. Select the first $b$ objects from that list. Call this set $S$.
3. If $\sum_{i \in S} c_i \leq k$, then the output of the algorithm is YES. Else, the output is NO.

Clearly, this algorithm is correct and runs in polynomial time w.r.t. to the instance. Hence, $Q \in \mathcal{P}$.

(b)
- Let $P$ be *SAT*.
- Let $Q$ be *3-SAT*.

We showed in class that both $P$ and $Q$ are NP-complete problems.

2. A *Hamiltonian cycle* in a graph $G = (V, E)$ is a simple cycle that contains all the vertices. A *Hamiltonian $s-t$ path* in a graph is a simple path from $s$ to $t$ that contains all of the vertices. The associated decision problems are:

   - *Hamiltonian Cycle* **Input:** $G = (V, E)$. **Question:** Does there exist a hamiltonian cycle in $G$?
   - *Hamiltonian Path* **Input:** $G = (V, E)$, $s, t \in V, s \neq t$. **Question:** Does there exist a hamiltonian $s$-$t$ path in $G$?

   (a) Given that *Hamiltonian Cycle* is NP-complete, prove that *Hamiltonian Path* is NP-complete.

   (b) Given that *Hamiltonian Cycle* is NP-complete, prove that the optimization version of the TSP problem is NP-hard.

## Solution:

(a) **Step 1: *Hamiltonian Path* is in NP.**

It is clear that *Hamiltonian Path* is in NP, the certificate to a YES answer is the path itself. Given a Hamiltonian path from $s$ to $t$. We only need to travel along it to check that in fact it visits every vertex once and that starts in $s$ and ends in $t$. This takes $O(|E|)$ time.

**Step 2: *Hamiltonian Cycle* $\leq_P$ *Hamiltonian Path*.**

Given an instance $G = (V, E)$ of *Hamiltonian Cycle*, we construct an instance of *Hamiltonian Path* as follows:

- Let $v \in V$, the we construct the graph $G' = (V', E')$, where:
  - $V' = V \setminus \{v\} \cup \{v_1, v_2\}$ (this takes $O(1)$).
  - $E' = (E \setminus \{\{v, u\} : \{v, u\} \in E\}) \cup \{\{v_1, u\} : \{v, u\} \in E\} \cup \{\{v_2, u\} : \{v, u\} \in E\}$ (this takes $O(|V|)$).
- The instance is: $G' = (V', E')$, $s = v_1$, $t = v_2$.

Notice the construction takes polynomial time w.r.t. the size of the instance.

Finally, the fact that a YES to an instance of *Hamiltonian Cycle* is equivalent to a YES to the associated *Hamiltonian Path* instance follows by noticing that there exists a Hamiltonian cycle of the form

$$v u_1 u_2 \dots u_{n-1} v$$

in $G$ if and only if there exists and Hamiltonian $v_1$-$v_2$ path in $G'$ of the form

$$v_1 u_1 u_2 \dots u_{n-1} v_2$$

in $G'$.

**Note:** we are denoting $n = |V|$ and the notation $u_0 u_1 \dots u_k$ represents the path/cycle that uses the edges $\{u_0, u_1\}\{u_1, u_2\} \dots \{u_{k-1}, u_k\}$ (in that order).

**Conclusion:** *Hamiltonian Path* is NP-complete.

(b) Given an instance $G = (V, E)$ of *Hamiltonian Cycle*, the following algorithm decides whether the answer to this instance is yes or no:

ALGORITHM:

1. Given $V = \{v_1, \ldots, v_n\}$, consider the cities $\{1, \ldots, n\}$.
2. Construct the objective function $c$ given by:

$$c_{ij} = \begin{cases} 0, & \{v_i, v_j\} \in E \\ 1, & \text{else.} \end{cases}$$

3. Solve the TSP instance given above. Let $\alpha$ be the cost of the optimal tour.
4. If $\alpha = 0$, then the output of the algorithm is YES. Else, the output is NO.

The algorithm is correct since the definition of the objective function implies that the optimal tour has cost equals to zero if and only if the tour only travels between pairs of cities associated to edges in $E$.

Notice that the algorithm only need to solve the TSP problem once and that every other step takes polynomial time. Therefore, this is a valid polynomial time reduction since if we were able to solve the optimization version of the TSP in polynomial time, we would also be able to decide *Hamiltonian Cycle* in polynomial time.

**Conclusion:** the optimization version of the TSP problem is NP-hard.

3. Given that the *node packing problem* is NP − complete, show that the following problems are also NP − complete:

(a) *Node cover*: **Input:** $G = (V, E)$, $k \in \mathbb{Z}_+$. **Question:** Does there exist a set $S \subseteq V$ of size at most $k$ such that every edge of $G$ is incident to a node of $S$?

(b) *Uncapacitated facility location*. **Input:** sets $M, N$ and integers k, $c_{ij}$, $f_j$ for $i \in M$, $j \in N$. **Question:** Is there a set $S \subseteq N$ such that $\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k$?

Recall that *Node packing problem* is: **Input:** $G = (V, E)$, $k \in \mathbb{Z}_+$. **Question:** Does there exist an independent set of size at least $k$ in $G$?

## Solution:

(a) **Step 1: *Node cover* is in NP.**

It is clear that *Node cover* is in NP, the certificate is the node cover itself. Verifying that the set of nodes is a cover can be done by checking that every edge is connected to a node in the given set. This takes $O(|E||V|)$ time.

**Step 2: *Node packing* $\leq_P$ *Node cover*.**

Given an instance $G = (V, E)$, $l \in \mathbb{Z}_+$ of *Node Packing*, we construct an instance of *Node cover* as follows:

- We construct the graph $G' = (V', E')$, where:
  - $V' = V$ (this takes $O(1)$).
  - $E' = E$ (this takes $O(1)$).

- We take $k = |V| - l$ (this takes $O(1)$).
- The instance is: $G' = (V', E'), k \in \mathbb{Z}_+$.

Notice the construction takes polynomial time w.r.t. the size of the instance.

Finally, the fact that a YES to an instance of *Node packing* is equivalent to a YES to the associated *Node cover* instance follows by noticing that a set $U \subseteq V$ is a node packing in $G$ if and only if no edge in $E$ has both end points in $U$, which is equivalent to say that every edge in $E$ has at least one end point in $V \setminus U$. Equivalently, this is saying that the set $V \setminus U$ is a node cover in $G'$.

**Conclusion:** *Node cover* is NP-complete.

(b) ***Step 1:*** **Uncapacitated facility location is in NP.**

It is clear that *Uncapacitated facility location* is in NP, because given $S \subseteq N$, we can verify in $O(|M||N| + |N|)$ if

$$\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k$$

**Step 2:** ***Node packing*** $\leq_P$ ***Uncapacitated facility location.***

Given an instance $G = (V, E)$, $l \in \mathbb{Z}_+$ of *Node Packing*, we construct an instance of *Uncapacitated facility location* as follows:

- $M = E$, $N = V$ (this takes $O(|E|)$).
- The objective

$$c_{ij} = \begin{cases} |V| + 1, & \text{if } j = uv, \text{ where } u, v \neq i \\ 0, & \text{otherwise.} \end{cases}.$$

  (This takes $O(|E|^2)$.)
- $k = |V| - l$ (this takes $O(1)$).
- $f_j = 1$, for all $j \in N$ (this takes $O(|V|)$).

Notice the construction takes polynomial time w.r.t. the size of the instance.

- **YES to *Node Packing*** $\Rightarrow$ **YES to *Uncapacitated facility location*:**
  If $U$ is a node packing, with $|U| \geq l$, then $S = V \setminus U$ is a vertex cover, hence for all $i \in M$:

$$\min\{c_{ij} : j \in S\} = 0.$$

  And

$$\sum_{j \in S} f_j = |S| = |V| - |U| \leq |V| - l \leq k.$$

  This implies

$$\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k.$$

  Thus, the set $S \subseteq N$ gives a YES answer to *Uncapacitated facility location*.
- **YES to *Uncapacitated facility location*** $\Rightarrow$ **YES to *Node Packing*:**

There exists $S \subseteq N$ such that

$$\sum_{i \in M} \min_{j \in S} c_{ij} + \sum_{j \in S} f_j \leq k.$$

By definition of the reduction from node packing, we have

(i) For all $i \in M$, $\min\{c_{ij} : j \in S\} \leq |V|$, which implies $\min\{c_{ij} : j \in S\} = 0$.

(ii) Let $U = V \setminus S$. By (i), $\sum_{j \in S} f_j = |S| = |V| - |U|$.

(iii) By (i), if $u, v \in U$, then we must have $\{u, v\} \notin E$.

(iv) By (ii), we have $|U| \geq l$.

This implies that the set $U \subseteq V$ is a node packing with $|U| \geq l$, which gives a YES answer to *Node Packing*.

**Conclusion:** *Uncapacitated facility location* is NP-complete.