NP-Completeness

Fabio A. González
fagonzalezo@unal.edu.co

Programación Matemática a Gran Escala
Maestria en Ingeniería de Sistemas
Universidad Nacional de Colombia

The Computational Complexity Theory studies problems and their intrinsic computational difficulty: given a problem, how much computing power and/or resources do we need in order to solve it?. From the algorithmic point of view, the Computational Complexity Theory gives lower bounds to the time complexity of the algorithms that solve some kinds of problems.

An important class of problems is the class *NP* (problems with deterministic polynomial time verification algorithms), Some problems in this class are known to be very difficult to solve. To date, nobody has found a better than exponential time algorithms to solve these problems, but the best lower bound that the researchers have been able to prove is polynomial time. A subclass of *NP* problems is called *NP-Complete*, and the most interesting thing is that if we are able to find a polynomial time algorithm to solve one problem in the *NP-Complete* class, we can build algorithms to solve all the problems in *NP* in polynomial time.

1 Preliminary Definitions

1.1 Problem, parameters, problem instance, solution

An abstract problem π is defined as a function (relation) from a set I of instances to a set S of problem solutions. An instance of the problem is obtained by assigning values to the parameters of the problem. Thus, describing the set of instances (possible inputs) and solutions (possible outputs) defines a problem.

$$\pi : I \rightarrow S$$

Generally the sets I and S are sets of binary strings, $I, S \subseteq \{0,1\}^*$.

1.2 Algorithm

An algorithm that solves a problem must give correct solutions to all instances of the problem using finite resources (time and space), otherwise, it is not considered as an algorithm. In other words, algorithm is a mapping from the set of all instances to the set of corresponding solutions.

1.3 Input length

Input length is the length of encoding of an instance of the problem. Generally it is measured in bits. The time and space complexity of an algorithm are described as functions of its length.

1.4 Worst-case time/space complexity

The worst-case complexity is a function from the positive integers to the positive reals:

$$\begin{array}{rccc} t & : & Z^+ & \longrightarrow & R^+ \\ & & n & \longmapsto & t(n) \end{array}$$

Where n is the length of an instance, and $t(n)$ is the maximum time (or space) required to get solution for any instance of length n. Of course, not all the instance of length n need same time/space.

1.5 Decision problems

A decision problem is a problem whose solution is either YES or NO. $p : I \rightarrow \{Yes, No\}$. For example, "Is graph G a connected graph?" is a decision problem.

1.6 Optimization problems

In a optimization problem we want to maximize or minimize a quantity. For instance, the maximum-flow and the minimum spanning tree are optimization problems.

We can construct a decision problem from an optimization problem, for instance, from the maximum-flow problem we can generate the following decision problem "Does the graph G have a flow greater than k?" where k is a constant.

2 Complexity classes

2.1 The complexity class *P*

P is the set of abstract problems that are solvable by polynomial-time algorithms. Formally, P is the class of abstract problems, which have an algorithm that solve it in time $t(n)$ and $t(n) \in O(n^k)$ for some constant k (this is equivalent to saying that $t(n)$ is a polynomial).

The problems that belong to this class are considered "efficiently solvable", clearly this is a subjective affirmation, however it is generally accepted.

An interesting property of the polynomial problem in P is that any composition of this problems are also in P.

2.2 The complexity class *Exp*

Exp is the class of problems that are solvable by exponential time algorithms. Formally, Exp is the class of abstract problems, which have an algorithm that solve it in time $t(n)$ and $t(n) \in O(2^{n^k})$ for some constant k.

Clearly the class P is contained in the class Exp, that is, Exp is a more general class. The problems contained in Exp but not in P, are generally considered hard problems.

2.3 Polynomial-time verification algorithms

Given an instance i of a decision problem and a certificate s, the verification algorithm determines (in polynomial time) with the help of the certificate whether the solution for the problem instance i is Yes. (note that it cannot verify a No answer).
Examples:

Problem	**Certificate**
Does the graph G have a odd-length cycle?	The nodes list of the cycle
Is a given integer number a composite number?	The list of the prime factors of the number.

The size of the certificate has to be polynomial in the size of the input, that is, if the length input is n the length of the certificate $|s| \in O(n^c)$ for some constant c.

2.4 The complexity classes *NP* and *Co-NP*

NP is the class of decision problems that have a polynomial-time verification algorithm. *Co-NP* is the class of decision problems for which the corresponding complement problem belongs to *NP*. A complement of a decision problem π, is a problem $\overline{\pi}$ such that $\forall i \in I$, $\overline{\pi}(i) = Yes$ if and only if $\pi(i) = No$.

For instance, the problem "Is a given integer number a composite number?" belongs to the class *NP*, and the problem "Is a given integer number a prime number?" belongs to the class *Co-NP*.

2.5 *NP-Complete* problems

NP-Complete is the set of decision problems that are as hard as any problem in *NP*, that is, it is possible to reduce any *NP* problem to a problem in *NP-Complete*. There is not proof that NP-complete problems are tractable or not.

To prove that a problem Π is *NP-Complete*, we need to show that:

1. $\Pi \in$*NP* and
2. a known *NP-Complete* problem Π' reduces to Π.

Some examples of *NP-Complete* problems:

- **SAT** (Satisfiability problem): Given a propositional logic formula, is the formula true for some combination of truth values of its variables?
- **3SAT**: Same as SAT, but each clause (in disjunctive normal form) has at most 3 variables.

- **Vertex Cover**: Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, is there a vertex cover of size k or less for G? , that is, a subset $V' \subseteq V$ such that $|V'| \leq k$ and, for each edge $(u, v) \in E$, at least one of u and v belongs to V'?
- **Hamiltonian Cycle**: Given a graph $G = (V, E)$, is there a cycle that includes all the vertices of G once and only once?
- **TSP** (Travel Salesman Problem): Given a graph $G = (V, E)$ and a constant C, is there a cycle that includes all the vertices of G once and only once and with total cost less or equal to C?

3 The hierarchy of complexity classes

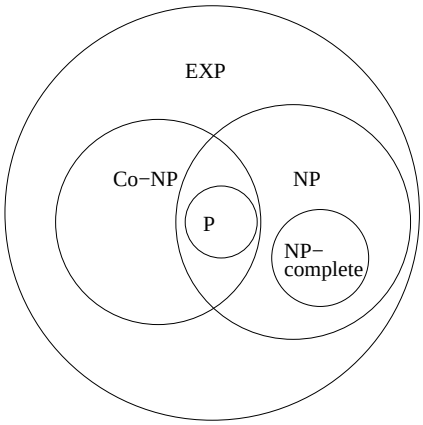

Figure 1: Complexity classes diagram

Figure 1 shows the known relationships between the complexity classes mentioned above. The only strict containment relationships ($\subset$) are $P \subset Exp$ and $NP\text{-}Complete \subset NP$, the others are not strict ($\subseteq$). The following are the major unsolved questions:

- *P=NP?*
- *NP=Co-NP?*
- *P=NP∩Co-NP?*

References

[1] *Class Notes COMP7713*, Giri Narasimhan and Fabio González, Fall 2000. *http://www.msci.memphis.edu/~giri/7713/*

[2] *"Handbook of theoretical computer science"*, Edited by Jan van Leeuwen, MIT Press 1990.

[3] *"Introduction to algorithms"*, Thomas Cormen , et al. MIT Press, 1990.