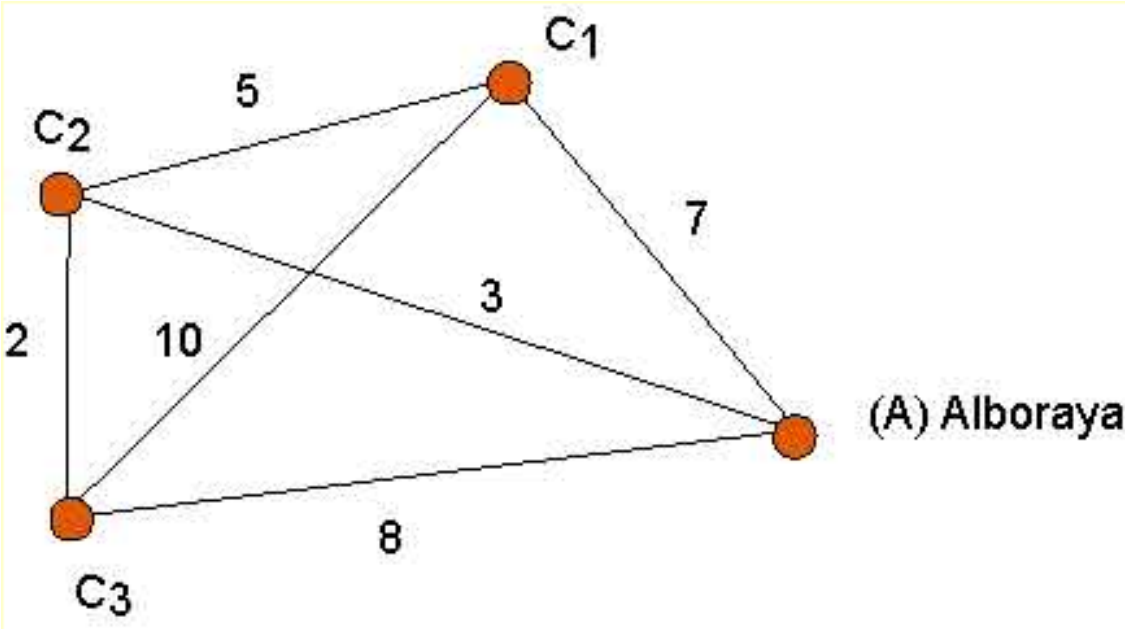
# TEMA 7. COMPLEXITY

## 1. Introduction.

If you know how to solve a problem, you can always find a solution. This statement seems obvious. But it is not true. As an example, we are going to study the traveling salesman problem: a traveling salesman wants to leave Alboraya, and visit the 50 most important cities in Spain, to finally return to Alboraya, with the intention of convincing some merchants to market their horchata. The problem that the traveler faces is finding the itinerary that involves traveling the minimum distance. How could the question be solved?

This problem is within the Graph Theory. It consists of "finding a Hamiltonian circuit in a graph". And there is not always such a circuit. This is a problem that remains open: no criteria are known to distinguish Hamiltonian graphs from those that are not. But if the graph is complete, it always exists. Let us suppose then that the graph is complete, that is to say that from any city (vertex) any other city can be reached, by means of a road that does not pass through any other vertex. This problem can be summarized as follows: They obtain all the possible itineraries that starting from Alboraya arrive at Aboraya, visiting the 50 cities.

Let's see an example for 3 cities. We construct the graph of the problem, as shown in the following figure: each pair of cities is joined by an edge, and the number on each reflects the cost in km. which implies going through it, one way or the other (it is an undirected graph).




A possible solution to the problem would be A-C1-C3-C2-A. In other words, leave Alboraya, go to C1, then C3, then C2, and finally return to Alboraya. In this case, the cost in kilometers would be: 7+10+2+3=22 km. How many possible paths with these characteristics are there? Exactly 3! = 6. These possibilities and their corresponding cost are given in the following table.

| hamiltonian Circuit | Cost |
|---|---|
| A-C1-C2-C3-A | 7 + 5 + 2 + 8 = 22 |
| A-C1-C3-C2-A | 7 + 10 + 2 + 3 = 22 |
| A-C2-C1-C3-A | 3 + 5 + 10 + 8 = 26 |
| A-C2-C3-C1-A | 3 + 2 + 10 + 7 = 22 |
| A-C3-C1-C2-A | 8 + 10 + 5 + 3 = 26 |
| A-C3-C2-C1-A | 8 + 2 + 5 + 7 = 22 |

We select the path that has the least cost. We have four possible solutions: one, two, four, and six. Any of them would solve the problem for us. What happens by applying this algorithm to our initial problem? Let's see: the total number of Hamiltonian circuits would be:

50! = 30414093201713378043612608166064768844377641568960512000000000000

Suppose a computer takes a second to obtain and evaluate each circuit: this would suppose that it would be working $9.644245688 \cdot 10^{56}$ years. And you would still have to find the Hamiltonian minimal cost circuit among that huge number of possible solutions! We know how to solve the problem, but we cannot find a solution, at least, by this method.

**2. Theory of the algorithmic complexity.**

The previous example leads us to study a question: "the difficulty of the problems". We can know how to solve a problem and cannot find a solution. There are problems that the computer can solve with acceptable computational time, and others that cannot. For example, obtaining the solutions of a polynomial equation of the third degree has a low computational cost. It is an easy problem. But obtaining a Hamiltonian cycle in a complete graph has a very high computational cost. It is a difficult problem.

In order to analyze this question we need a suitable language (or notation). We are going to establish the rules that allow us to talk about these problems. When we talk about a problem we are referring to a task that needs to be done. Let us make this more specifically: a problem is a general question, posed in terms of open parameters, to which a solution must be found. In this sense, the word problem is actually applied to a collection of specific problems. By giving values to the parameters, we say that we have an instance of the problem.

Let's see some simple example

Example 1.

- One problem: solving the equation $ax^2+bx+c=0$. Actually, what we want is to solve all the equations of second degree.
- An instance of the problem: $3x^2+9x+10=0$

Example 2.

- One problem: finding a Hamiltonian circuit in a complete graph, at minimum cost.
- An instance of the problem: find a Hamiltonian circuit of minimum cost, in the graph of the previous example.

An algorithm is a step-by-step procedure that solves a problem. We say that an algorithm solves a problem when it is capable of solving any instance of the problem. As we have said, it may happen that we are able to solve an instance of a problem, but that we are not able to solve the problem. On the one hand, the efficiency of an algorithm must be considered. We assume that the most efficient algorithm is the fastest. It can be expected that the time it takes for an algorithm to resolve an instance of a problem depends on its size. The problem is that the time that an algorithm takes to solve a problem depends on the power of the processor of the computer in which we are working.

To avoid this dependency we define the computation time in solving a problem: given a problem p and an algorithm A (which is capable of solving an instance I of p) we define computation time spent by A in solving I, as the number of elementary steps (elementary arithmetic operations and read and write operations) performed by the computer to solve I. With this information, the computation time of an algorithm A to solve a problem can be defined: for a problem p and an algorithm A (that solves any instance I of p of size n) we call the computation time of the algorithm the maximum computation time required by the algorithm to solve any instance I, of size less than or equal to n. We will call this time using the notation $f_A(n)$.

Under these conditions the algorithms are classified into two large groups: they will be polynomial time algorithms when they solve a problem in polynomial time, that is, if there is a constant c such that $|f_A(n)|<=c|p(n)|$, where p(n) is a polynomial in n. In any other case, we will say that A is an exponential algorithm.

What does this mean? Polynomial algorithms take time to solve a problem that can be delimited by a polynomial p(n), which depends on the size n of the problem. In this way, as the size of the problem increases, the time taken by the algorithm to solve the problem grows, but slowly (polynomically), since the growth rate of a polynomial is slow (the derivatives help us to know that speed). In contrast, in exponential time algorithms as the size of the problem grows, the resolution time grows very quickly (exponentially). The polynomial algorithms are the most efficient, and are, therefore, those that we must look for to solve a problem. Exponential algorithms are not capable of solving a problem for large sizes of the problem instance. Example 1 above is a problem that can be solved in polynomial time. Instead, Example 2 above is an exponential problem.

## 3. Classification of the problems according to their algorithmic complexity.

Finally, it remains to classify the problems according to their algorithmic complexity. This classification is important because it will allow us to know what type of algorithm we have to look for to solve it. Our goal is to distinguish those problems that can be efficiently solved from those where this is not possible. For this classification we need a previous definition.

Decision problem. It is a problem that only has two possible answers: yes (or true), no (or false). Let's see some examples of this type of problem:

> Example 3. The problem, does an equation have solutions contained in a closed interval [a, b]? There are only two answers: yes or no.
> Example 4. Does the problem contain any graph some cycle? There are only two answers: yes or no
> Example 5. Does the problem contain any graph some Hamiltonian cycle? There are only 2 answers: yes or no.

These three problems have a very different character: polynomial algorithms are available for the first and the second, but not for the third. Furthermore, it is not even known whether a polynomial algorithm can be found. A decision problem defines a set X of cases in which the correct answer is yes (yes-cases). The rest of the cases are no-cases. We will say that a correct algorithm that solves the decision problem accepts the yes case and rejects the no cases. This is what will allow us to define classes P and NP.

**4. P and NP classes.**

Class P: we say that a problem belongs to class P, if it is a decision problem that can be solved using a polynomial algorithm. Therefore, these problems are treatable. Those problems for which the best known solution is more complex than polynomial, are said to be intractable. It sometimes happens that a polynomial algorithm is available to check whether a possible solution is valid or not, but such algorithms are not available to solve any instance of the problem. These problems form the NP class. For example, to check whether a cycle of a graph is Hamiltonian or not, a polynomial algorithm is available. But to find Hamiltonian cycles in a graph, the algorithm that we have at the moment is exponential. The problem in Example 5 is of the NP class.

It is evident that class P problems are included in NP class problems. But it is not known if the inclusion is strict. This problem (knowing if P=NP) is one of the most important within the algorithmic complexity, and has remained open since 1971. But within the problems of the NP class, there is still a second classification. The NP-complete problems, within the NP problems, are the most difficult from the point of view of computational complexity. The problem of finding a Hamiltonian cycle in any graph is NP-complete. Within NP-complete problems are NP-hard problems: they are NP-complete problems, which are not necessarily a decision problem. The problem of school schedules is NP-hard.

Some theorems to settle the question:

> Theorem1: if a problem is NP and there is an algorithm that solves it in polynomial time, then P=NP.
> Theorem2: if a problem is NP-complete and there is an algorithm that solves it in polynomial time, then P=NP.
> Theorem3: if a problem is NP-hard and there is an algorithm that solves it in polynomial time, then P=NP.

What is all this for? Well, to know what type of algorithm we should look for a new problem depending on whether it has been classified in class P, or in class NP-hard. If a

problem is NP-hard, and we assume that P and NP are different classes, we cannot expect to find a polynomial algorithm to solve it. Non-deterministic algorithms called heuristics will be sought for these problems, which, although they are not capable of solving the problem by giving the optimal solution, find good solutions. They apply heuristics to obtain hypothetical solutions that are disregarded (or accepted) at a polynomial rate.

What if P equals NP? Then it would make sense to search for NP-hard and NP-complete problems for exact algorithms, which will solve the problem in polynomial time. But, although unproven, the scientific community today is almost certain that P and NP are different. Summarizing: to approach the study of a new problem using an algorithm (computer), it is convenient to determine if it is P or NP-hard. In case it is of class P, we will look for an exact algorithm that solves it, and that may be polynomial. If it is of the NP-hard class, we will not claim an exact algorithm, since then computational times skyrocket. We will look for a heuristic that in an acceptable computational time is capable of giving a solution as close as possible to the optimal solution.