2. Complexity of problems and algorithms

This chapter presents an introduction to the theory of complexity. Decision problems, optimisation problems, couting problems and enumeration problems are defined, and complexity classes associated to these problems are introduced. These classes aim at qualifying the difficulty of solving problems. We first start with some considerations on the complexity of solution algorithms.

2.1 Complexity of algorithms

The complexity of an algorithm lies in estimating its processing cost in time (time complexity) or in the required space memory (spatial complexity). Set apart for certain particular algorithms, as for example dynamic programming algorithms which usually take up a lot of memory space, spatial complexity has been less considered than time complexity. In both cases it is possible to propose a *theoretical* complexity and a *practical* complexity. Theoretical complexity reflects an independent estimate on the machine which processes the algorithm. It is less accurate than the practical complexity which enables us to calculate the cost of the algorithm for a given computer. For the latter case, time complexity is obtained using an estimation of the calculation time for each instruction of the program. The advantage of theoretical complexity is that it provides an estimation independent of the calculation time for the machine.

In the remainder of this section we use the term *complexity* to refer to the time complexity of an algorithm. This complexity is established by calculating the number of iterations done by the algorithm during its processing. The number of iterations depends on the size of the data, noted $Length$, and possibly the magnitude of the largest element, noted Max, belonging to these data. If the number of iterations is bounded by a polynomial function of $Length$ then the algorithm is of polynomial complexity. If this function is limited by a polynomial of Max and $Length$, then we say that the algorithm is of pseudo-polynomial complexity. In other cases the algorithm is said to be of exponential complexity.

More precisely, we can distinguish minimal, average, and maximal complexities in order to translate complexity in the best case, the average case or in the worst case respectively. These latter two actually are interesting and the easiest to calculate is maximal complexity. On the other hand, average complexity requires a statistical analysis of the processing of the algorithm by function of the input data.

Example.
We can illustrate these notions by the example presented in figure 2.1. The maximum number of iterations is equal to n and the minimum number to 1. The average complexity itself depends on the probability p that the element is found in a given position. We suppose that this probability follows a uniform law, *i.e.* $p = \frac{1}{n}$. Thus, the average complexity is equal to $p(1+2+3+...+n) = \frac{n(n+1)}{2n} = \frac{n+1}{2}$. We notice that the calculation is only valid if we are sure that the element belongs to the list. In the opposite case, the calculation of the average complexity is even more complicated as it causes the law of generation of the elements of the list to intervene.

Search for an element belonging to a list

/* *elt* the searched element */
/* n is the list size */
/* *list* is the list of elements */
/* We assume that *elt* belongs to the list */
$i = 0$;
While ($elt \neq list[i]$) Do
  | $i = i + 1$;
End While;

Fig. 2.1. Search for an element in a non sorted list

To calculate the complexity of an algorithm, it is possible either to count the number of iterations, as we have done in the above example, or to break up the algorithm into sub-algorithms of known complexity. In this latter case, we can multiply or add the complexities according to the structure of the program. By using the above example, we can propose an algorithm which searches k elements in a list. Its average and maximal complexities are then of the order of $k \times n$. In the case of spatial complexity, the calculation cannot be performed on the algorithm itself -we cannot count the number of iterations- but rather on the data it uses.

The theoretical complexity of an algorithm is usually a function of Max, of *Length* and of addition and multiplying constants. Very often, we resume this complexity by the expression of the term which gives its asymptotic value. For