Computational complexity

H. R. Alvarez A., Ph. D.
¿What is it?

Part of the computational theory that studies the resources required during computation to solve a problem. The commonly studied resources are:

- time (number of execution steps of an algorithm for solving a problem)
- space (amount of memory used to solve a problem).
Combinatory complexity

- It is based on the number of components of a system, or the number of possible combinations to be performed when making a decision.
- It is a function of both the variables and the functions that govern or shape the system.
Algorithms and problem solving

- From the Greek and Latin, “dixit algorithmus”, originally from Persian mathematician Al-Khwarizmi.

- Informally, an algorithm is a well-defined computational procedure that takes a set of values (inputs) and produces a set of values (outputs) using a sequence of computational steps in the transformation.

- Prescribed set of well-defined, finite and ordered rules or instructions, that enables a solution process through successive steps that generate no doubt who should perform this activity.

H. R. Alvarez A., Ph. D.
Types of algorithms

Ordering algorithms:
- Let the input be a sequence of $n$ numbers $(a_1, a_2, \ldots, a_n)$
- The output will be the permutation or ordering $(a'_1, a'_2, \ldots, a'_n)$, such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Search algorithms:
- Let the input be a sequence of $n$ numbers $(a_1, a_2, \ldots, a_n)$
- The output will be a number $a^k$ such that $a^k \supset \{\text{characteristics}\}$
Design and solution techniques

- **Greedy algorithms**: select the most promising elements of the set of candidates to find a solution. In most cases the solution is not optimal.

- **Parallel algorithms**: allow dividing a problem into sub problems so that they can run simultaneously on multiple processors.

- **Probabilistic algorithms**: some of the steps of such algorithms are based on pseudo-random values.

- **Deterministic algorithms**: the behavior of the algorithm is sequential: each step of the algorithm has only one preceding step and another successor step.
Design and solution techniques

- **Non-deterministic algorithms**: the behavior of the algorithm is a tree and each step of the algorithm can branch to any number of immediately following steps, plus all the branches are executed simultaneously.

- **Divide and conquer**: divides the problem into disjoint subsets obtaining a solution for each subset. It then unites them, achieving a solution to the whole problem.

- **Meta heuristics**: It finds suboptimal or approximate solutions to problems based on prior knowledge (sometimes called experience).
Design and solution techniques

- **Dynamic programming**: tries to solve a problem through different sequential steps, tracking back possible solutions. It will examine the previously solved subproblems and will combine their solutions to give the best solution for the given problem.

- **Branch and bound**: Based on the construction of the solutions to a problem through an implicit tree that runs in a controlled manner by finding the best solutions.
Properties (no for parallell algorithms)

- **Sequential time.** An algorithm runs in discretized step by step time, thus defining a sequence of "computational" states for each valid entry.

- **Abstract state.** Each computational state can be formally described using a first-order structure and each algorithm is independent of its implementation.

- **Bounded exploration.** The transition from one state to the next is completely determined by a fixed and finite description; that is, between each state and the next you can only take into account a fixed and limited amount of possible current states.
The problem

- The resulting problem of a mathematical model has three elements:
  - The problem: the ultimate question
  - Elements: a list of parameters, variables and relationships, characteristic of the solution
  - Instances: parameter values
Type of problems

- **Tractables or decidable problems**: there are algorithms capable of optimally solving them.

- **Undecidable or not tractables problems**: there are no algorithms that can optimally solve them.
Efficiency of an algorithms

- The notation that describes the behavior, as a function of time, of the execution of an algorithms, is asymptotically approximate.

\[ \Theta(f(n)) = \{ f(n) \mid \exists c_1, c_2, n_0 / 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \ \forall \ n \geq n_0 \} \]

\[ O(f(n)) = \{ f(n) \mid \exists c_1, n_0 / 0 \leq c_1 g(n) \leq f(n) \ \forall \ n \geq n_0 \} \]

\[ \Omega(f(n)) = \{ f(n) \mid \exists c_2, n_0 / 0 \leq f(n) \leq c_2 g(n) \ \forall \ n \geq n_0 \} \]
Polynomial problems

- One problem is Polynomial (P) if there is a deterministic polynomial time algorithm to solve it.
  - When the running time of an algorithm is less than a certain value determined in terms of the length of the input variable (n) a problem can be solved in polynomial time.
- An algorithm is efficient if a problem can be solved such that the number of steps to resolve grows polynomially depending on their size.
- Can be approximated to a solution in terms of $n^k$
Non-Polynomial (NP) Problem

- If there is no deterministic polynomial algorithm to solve it.
- A special case are the intractable problems, which include:
  - Consistently intractable: Those that are so difficult that not even a non-polynomial time algorithm can solve it.
  - Seemingly intractable: The problem is so difficult that an exponential time is required to find a solution. The solution is so large that can not be expressed as a polynomial function of the input.
- Within the class NP "difficult" NP-complete problems as defined If there is no deterministic polynomial algorithm to solve.
Efficiency of some algorithms

- Simplex $O(n^k)$
- Interior point (Karmakar and others) $O(n \log(n))$
- Integer programing NP-Complete
  - Branch and bound $O(k^n)$
- TSP NP-Complete