Computational complexity theory

Abdelwahid Benslimane

wahid.benslimane@gmail.com

The computational complexity theory looks at the impact of increasing the size of an algorithm's inputs (n) on computation time and memory requirements. For a given problem, we always perform a pessimistic evaluation.

Let's start with a few definitions…

Big O domination relationship

A function f is said to be dominated by g in the neighborhood of $+\infty$, and we write $f = O(g)$ (Landau notation) if, and only if there are constants N and C such that:

$\forall x \in R, x > N \rightarrow ||f(x)|| \leq C||g(x)||$

Asymptotic equivalence

A function f f is said to be asymptotically equivalent to g in the neighborhood of $+\infty$, and we write $f = \Theta(g)$ if, and only if $f = O(g)$ and $g = O(f)$. This is an equivalence relationship.

Preponderance relationship

A function f is said to be negligible in front of g (or g is preponderant in front of f) in the neighborhood of $+\infty$, and we write $f = o(g)$ if, and only if:

$$\forall \epsilon \in R_+^*, \exists N \in R, \forall x \in R, x > N \rightarrow ||f(x)|| \leq \epsilon||g(x)||$$

Function equivalence

The binary relationship $f \sim g$ in the neighborhood of $+\infty$, which means that $(f - g)$ is negligible in front of g in the neighborhood of $+\infty$, is an equivalence relationship. For example, $3x^2 + 2x + 1 \sim 3x^2$ in the neighborhood of $+\infty$. However, the multiplying factor is of little importance. Instead, we write $3x^2 + 2x + 1 = \Theta(x^2)$ even if we lose precision. As computer scientists are mainly interested in the upper bound of their algorithms' complexity, they write $3x^2 + 2x + 1 = O(x^2)$ instead, which is even less precise.

Some notations

- $\Theta(1)$ = constant complexity
- $\Theta\big(log(n)\big)$ = logarithmic complexity
- $\Theta(log(n)^c), c > 1$ = poly-logarithmic complexity
- $\Theta(n)$ = linear complexity
- $\Theta\big(n.log(n)\big)$ = linearithmic complexity
- $\Theta(n^2)$ = quadratic complexity
- $\Theta(n^3)$ = cubic complexity
- $\Theta(n^c), c > 1$ = polynomial complexity
- $\Theta(c^n), c > 1$ = exponential complexity
- $\Theta(n!)$ = factorial complexity

For example, for $n = 10^6$ and a computer capable of performing 10^6operations per second, an algorithm of cubic complexity $\Theta(n^3)$, thus requiring 10^{18} operations, will take 32,000 years to execute, whereas an algorithm of quadratic complexity $\Theta(n^2)$ (10^{12} opérations) will take “only” 11.6 days.

Running an exponential algorithm would be insane. For example, for an algorithm of complexity $\Theta(2^n)$ (10^{301030} operations), the execution time would be 10^{301006} times the estimated age of the universe.

A function that grows faster than any polynomial is called super-polynomial, and a function that grows slower than any exponential is called sub-exponential. For example, $n^{log(n)}$ is both super-polynomial and sub-exponential.

Problems that cannot be solved in polynomial time are called hard problems because calculating their solution quickly becomes impossible even for relatively small values of n, as in the case of super-polynomial or exponential algorithms. Alan Turing demonstrated that certain problems were undecidable, i.e. that it was impossible to conceive an algorithm to solve them.

TIME/SPACE duality

The computational complexity of an algorithm is measured by 2 parameters: complexity in time (TIME) and complexity in computation space/memory requirements (SPACE). There is a duality between computation time and memory space used. For example, to avoid calculating 2^{160} values, they can be pre-calculated and stored, but this will require sufficient memory.

Complexity classes

Complexity theory classifies problems according to the algorithms needed to solve them on an abstract model called a Turing machine. This is a finite-state machine with read/write memory in the form of an infinite ribbon.

Problems can be classified into complexity classes depending on the complexity of their solution. These classes and their hierarchy are given below:

$$P \subset NP \subset PSPACE \subset EXPTIME$$

The P class contains all problems that can be solved in polynomial time.

The NP class contains all problems that can be solved in nondeterministic polynomial time or in polynomial time on a non-deterministic Turing machine, which has the particularity of having several possible transitions on each state, which it explores in parallel. We can imagine it duplicating itself. More concretely, this means that such a machine can check in polynomial time whether a solution proposed for a given NP class problem is indeed a solution.

the $PSPACE$ class includes problems that can be solved by an algorithm using a memory space whose size is at most a polynomial function of the input size.

The $EXPTIME$ class includes problems that can be solved in exponential time.

Some specific problems in NP are as difficult as any other, i.e. all other NP problems are reducible to them in polynomial time. These are called NP-hard or NP-complete problems. Solving just one of them would mean proving that $P = NP$, which is a central question in complexity theory.

An example of an NP-complete problem is that of the travelling salesman. A travelling salesman must visit n different cities and has only one full tank of gas. He can therefore only cover a maximum distance. Is there an itinerary that would allow him to visit each city once and only once with this full tank of gas? This is a generalization of the Hamiltonian circuit problem from graph theory.

EXPTIME-complete problems are those that cannot be solved in polynomial time because it has been shown that some problems really do require exponential time to be solved, or in other words that $EXPTIME \neq P$.

$PSPACE$-complete problems have the following property: if any of them is in NP, then $PSPACE = NP$, and if any of them is in P, then $PSPACE = P$. This has not yet been proved, but it is strongly assumed that $PSPACE \neq NP$, or, in other words, that there are problems in $PSPACE$ that we think are more difficult than NP.